GT.M

Administration and Operations Guide

V7.0-004

Empowering
the Financial World

FIS

GT.M Administration and Operations Guide

Revision History		
Revision V7.0-004	20 September 2022	Updated the following chapters for V7.0-004: • Chapter 3: "Basic Operations" (page 13) • Chapter 5: "General Database Management" (page 88) • Chapter 13: "GT.CM Client/Server" (page 478)
Revision V7.0-003	24 June 2022	Updated the following chapters for V7.0-003: • Chapter 3: "Basic Operations" (page 13) • Chapter 5: "General Database Management" (page 88) • Chapter 7: "Database Replication" (page 230) • Chapter 12: "Database Encryption" (page 441)
Revision V7.0-002	29 March 2022	Updated the following chapters for V7.0-002: • Chapter 3: "Basic Operations" (page 13) • Chapter 5: "General Database Management" (page 88) • Chapter 6: "GT.M Journaling" (page 181) • Chapter 7: "Database Replication" (page 230) • Chapter 10: "Database Structure Editor" (page 352) • Chapter 12: "Database Encryption" (page 441)
Revision V7.0-001	24 November 2021	Updated the following chapters for V7.0-001: • Chapter 2: "Installing GT.M" (page 4)

		• Chapter 3: "Basic Operations" (page 13)
		• Chapter 4: "Global Directory Editor" (page 45)
		• Chapter 5: "General Database Management" (page 88)
		• Chapter 6: "GT.M Journaling" (page 181)
		• Chapter 7: "Database Replication" (page 230)
		• Chapter 10: "Database Structure Editor" (page 352)
		• Chapter 11: "Maintaining Database Integrity" (page 400)
		• Chapter 12: "Database Encryption" (page 441)
Revision V7.0-000	12 February 2021	Updated the following chapters for V7.0-000:
		• Chapter 3: "Basic Operations" (page 13)
		• Chapter 4: "Global Directory Editor" (page 45)
		• Chapter 5: "General Database Management" (page 88)
		• Chapter 6: "GT.M Journaling" (page 181)
		• Chapter 7: "Database Replication" (page 230)
		• Chapter 9: "GT.M Database Structure(GDS)" (page 338)
		• Chapter 10: "Database Structure Editor" (page 352)
		• Chapter 11: "Maintaining Database Integrity" (page 400)
Revision V6.3-014	06 October 2020	Updated the following chapters for V6.3-014:
		• Chapter 3: "Basic Operations" (page 13)
		• Chapter 5: "General Database Management" (page 88)
		• Chapter 7: "Database Replication" (page 230)
		• Chapter 10: "Database Structure Editor" (page 352)
		• Chapter 13: "GT.CM Client/Server" (page 478)
Revision V6.3-013	30 June 2020	Updated the following chapters for V6.3-013:
		• Chapter 3: "Basic Operations" (page 13)

		• Chapter 4: "Global Directory Editor" (page 45)
		• Chapter 5: "General Database Management" (page 88)
		• Chapter 6: "GT.M Journaling" (page 181)
		• Chapter 7: "Database Replication" (page 230)
		• Chapter 9: "GT.M Database Structure(GDS)" (page 338)
		• Chapter 10: "Database Structure Editor" (page 352)
		• Chapter 11: "Maintaining Database Integrity" (page 400)
Revision V6.3-012	08 April 2020	Updated the following chapters for V6.3-012: • Chapter 1: "About GT.M" (page 1) • Chapter 3: "Basic Operations" (page 13) • Chapter 4: "Global Directory Editor" (page 45) • Chapter 5: "General Database Management" (page 88) • Chapter 6: "GT.M Journaling" (page 181)
Revision V6.3-011	20 December 2019	Updated the following chapters for V6.3-011: • Chapter 3: "Basic Operations" (page 13) • Chapter 4: "Global Directory Editor" (page 45) • Chapter 5: "General Database Management" (page 88) • Chapter 7: "Database Replication" (page 230)
Revision V6.3-010	31 October 2019	Updated the following chapters for V6.3-010: • Chapter 2: "Installing GT.M" (page 4) • Chapter 3: "Basic Operations" (page 13) • Chapter 4: "Global Directory Editor" (page 45) • Chapter 5: "General Database Management" (page 88) • Chapter 6: "GT.M Journaling" (page 181) • Chapter 8: "M Lock Utility (LKE)" (page 324)

		• Chapter 11: "Maintaining Database Integrity" (page 400)
Revision V6.3-009	27 June 2019	Updated the following chapters for V6.3-009: • Chapter 3: "Basic Operations" (page 13) • Chapter 5: "General Database Management" (page 88) • Chapter 6: "GT.M Journaling" (page 181) • Chapter 7: "Database Replication" (page 230)
Revision V6.3-008	24 April 2019	Updated the following chapters for V6.3-008: • Chapter 3: "Basic Operations" (page 13) • Chapter 5: "General Database Management" (page 88) • Chapter 7: "Database Replication" (page 230) • Chapter 9: "GT.M Database Structure(GDS)" (page 338)
Revision V6.3-007	04 February 2019	Updated the following chapters for V6.3-007: • Chapter 12: "Database Encryption" (page 441) • Chapter 7: "Database Replication" (page 230) • Chapter 10: "Database Structure Editor" (page 352) • Chapter 9: "GT.M Database Structure(GDS)" (page 338) • Chapter 5: "General Database Management" (page 88) • Chapter 4: "Global Directory Editor" (page 45) • Chapter 6: "GT.M Journaling" (page 181) • Chapter 2: "Installing GT.M" (page 4)
Revision V6.3-006	26 October 2018	Updated the following chapters for V6.3-006: • Chapter 7: "Database Replication" (page 230) • Chapter 9: "GT.M Database Structure(GDS)" (page 338) • Chapter 5: "General Database Management" (page 88) • Chapter 4: "Global Directory Editor" (page 45)

		• Chapter 6: "GT.M Journaling" (page 181) • Chapter 2: "Installing GT.M" (page 4) • Chapter 8: "M Lock Utility (LKE)" (page 324) • Chapter 11: "Maintaining Database Integrity" (page 400)
Revision V6.3-005	03 July 2018	Updated the following chapters for V6.3-005: • Chapter 12: "Database Encryption" (page 441) • Chapter 7: "Database Replication" (page 230) • Chapter 9: "GT.M Database Structure(GDS)" (page 338) • Chapter 5: "General Database Management" (page 88) • Chapter 4: "Global Directory Editor" (page 45) • Chapter 2: "Installing GT.M" (page 4) • Chapter 8: "M Lock Utility (LKE)" (page 324) • Chapter 11: "Maintaining Database Integrity" (page 400)
Revision V6.3-004	23 March 2018	Updated the following chapters for V6.3-004: • Chapter 12: "Database Encryption" (page 441) • Chapter 7: "Database Replication" (page 230) • Chapter 5: "General Database Management" (page 88) • Chapter 4: "Global Directory Editor" (page 45) • Chapter 13: "GT.CM Client/Server" (page 478) • Chapter 6: "GT.M Journaling" (page 181) • Chapter 2: "Installing GT.M" (page 4) • Chapter 11: "Maintaining Database Integrity" (page 400)
Revision V6.3-003	12 December 2017	Updated the following chapters for V6.3-003: • Chapter 7: "Database Replication" (page 230)

		• Chapter 10: "Database Structure Editor" (page 352)
		• Chapter 5: "General Database Management" (page 88)
		• Chapter 4: "Global Directory Editor" (page 45)
		• Chapter 13: "GT.CM Client/Server" (page 478)
		• Chapter 6: "GT.M Journaling" (page 181)
		• Chapter 2: "Installing GT.M" (page 4)
Revision V6.3-002	22 August 2017	Updated the following chapters for V6.3-002:
		• Chapter 7: "Database Replication" (page 230)
		• Chapter 10: "Database Structure Editor" (page 352)
		• Chapter 5: "General Database Management" (page 88)
		• Chapter 4: "Global Directory Editor" (page 45)
		• Chapter 13: "GT.CM Client/Server" (page 478)
		• Chapter 6: "GT.M Journaling" (page 181)
		• Chapter 2: "Installing GT.M" (page 4)
Revision V6.3-001	20 March 2017	Updated the following chapters for V6.3-001:
		• Chapter 1: "About GT.M" (page 1)
		• Chapter 12: "Database Encryption" (page 441)
		• Chapter 7: "Database Replication" (page 230)
		• Chapter 10: "Database Structure Editor" (page 352)
		• Chapter 9: "GT.M Database Structure(GDS)" (page 338)
		• Chapter 5: "General Database Management" (page 88)
		• Chapter 4: "Global Directory Editor" (page 45)
		• Chapter 13: "GT.CM Client/Server" (page 478)
		• Chapter 6: "GT.M Journaling" (page 181)
		• Chapter 2: "Installing GT.M" (page 4)

		• Chapter 11: "Maintaining Database Integrity" (page 400)
Revision V6.2-001	27 February 2015	• Updated Chapter 2: "Installing GT.M" (page 4), Chapter 3: "Basic Operations" (page 13), Chapter 5: "General Database Management" (page 88), Chapter 6: "GT.M Journaling" (page 181), Chapter 7: "Database Replication" (page 230), Chapter 10: "Database Structure Editor" (page 352), Appendix D: "GT.M Security Philosophy" (page 496), and Appendix B: "Monitoring GT.M " (page 490) for V6.2-001 and V6.2-000. For chapter specific changes, refer to the chapter-level revision history.
Revision V6.1-000/1	04 September 2014	• In Chapter 2: "Installing GT.M" (page 4), added a new section called "Compiling the Reference Implementation Plugin " (page 10). • In Chapter 7: "Database Replication" (page 230) and under the "Procedures" (page 259) section, corrected the downloadable scripts example (msr_proc2.tar.gz) for setting up an A→P replication configuration.
Revision V6.1-000	01 August 2014	• Updated Chapter 3: "Basic Operations" (page 13), Chapter 4: "Global Directory Editor" (page 45), Chapter 5: "General Database Management" (page 88), and Chapter 7: "Database Replication" (page 230) for V6.1-000. For chapter specific changes, refer to the chapter-level revision history.
Revision V6.0-003/1	19 February 2014	• Added a new appendix called Appendix F: "Packaging GT.M Applications" (page 509). • In "REORG " (page 136), added a caution note on running successive REORGs. • In "Environment Variables" (page 19), improved the description of the gtm_nocenable environment variable. • In "File Header Data Elements " [339], updated the descriptions for V6.0-003.
Revision V6.0-003	27 January 2014	• In Chapter 3: "Basic Operations" (page 13), added the -help and -? command line options for the gtm script and the descriptions of gtm_ipv4_only and gtm_etrap environment variables. • In Chapter 4: "Global Directory Editor" (page 45), corrected the syntax of [NO]STDNULLCOLL and -COLLATION_DEFAULT and added a paragraph about what happens when database file with automatic extension disabled (EXTENSION_COUNT=0) starts to get full.

		• In Chapter 5: "General Database Management" (page 88), added the -MUTEX_SLOTs qualifier for MUPIP SET and -OVERRIDE qualifier for MUPIP RUNDOWN.
		• In Chapter 7: "Database Replication" (page 230), added downloadable script examples for A→B and A→P replication scenarios, information about the ACTIVE_REQUESTED mode, PASSIVE_REQUESTED mode, and IPv6 support.
		• In Chapter 9: "GT.M Database Structure(GDS)" (page 338),corrected the description of block header fields.
Revision V6.0-001/2	10 April 2013	• In Chapter 5: "General Database Management" (page 88), added a planning/preparation checklist that can be used before starting a MUPIP BACKUP.
		• In Appendix E: "GTMPCAT - GT.M Process/ Core Analysis Tool " (page 504), added information about the cmdfile option.
Revision V6.0-001/1	22 March 2013	• In Chapter 5: "General Database Management" (page 88), Chapter 6: "GT.M Journaling" (page 181), and Chapter 7: "Database Replication" (page 230), improved the formatting all command syntaxes.
		• In "SET Object Identifying Qualifiers " (page 195), added the descriptions of -repl_state={on|off} and -dbfilename=filename qualifiers of -jnlfile.
Revision V6.0-001	27 February 2013	• Added Appendix E: "GTMPCAT - GT.M Process/Core Analysis Tool " (page 504). gtmpcat is a diagnostic tool for FIS to provide GT.M support.
		• Updated for V6.0-001. For chapter-specific revisions, refer to Chapter 3: "Basic Operations" [13], Chapter 4: "Global Directory Editor" (page 45), Chapter 5: "General Database Management" (page 88), Chapter 6: "GT.M Journaling" (page 181), Chapter 7: "Database Replication" (page 230), and Chapter 10: "Database Structure Editor" (page 352).
Revision V6.0-000/1	21 November 2012	• Updated for V6.0-000. For chapter-specific revisions, refer to Chapter 3: "Basic Operations" [13], Chapter 4: "Global Directory Editor" (page 45), Chapter 6: "GT.M Journaling" (page 181), Chapter 10: "Database Structure Editor" (page 352), and Appendix B: "Monitoring GT.M " (page 490).

Revision V6.0-000	19 October 2012	• In Chapter 4: "Global Directory Editor" (page 45), added the description of the -INST_FREE_ON_ERROR region qualifier.
		• In Chapter 5: "General Database Management" (page 88), added the description of the -INST_FREE_ON_ERROR qualifier of MUPIP SET.
		• In Chapter 7: "Database Replication" (page 230), added the description of the -FREEZE qualifier of MUPIP REPLICATE -SOURCE.
		• In Chapter 3: "Basic Operations" (page 13) added gtm_custom_errors to the list of "Environment Variables" [19].

Table of Contents

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 andmanage 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. If you are looking for a GT.M command or a qualifier(s) that applies to a specific command, use the Command and Qualifier summary table at the end of the chapters.

There are two choices for the HTML version-plain HTML or webhelp. The left frame of the plain HTML format includes a Google custom search engine for searching across the manual. It requires an Internet connection and a Javascript-enabled browser.

The webhelp format includes a search facility that works even when you do not have an active Internet connection.

Whether you are a first-time user or an expert user, one of the fastest way to find information is to "Search" for it. Just type the keyword you are looking and press the Search button. With the Google custom search feature of the plain HTML format, you can search for an exact phrase by enclosing your search term within quotes. You can also combine keywords and exact phrases to restrict your search. Once you get your search result, use the browser find facility (ctrl+f in most browsers) to find the specific information. The search facility of the webhelp format takes only keywords. If there are more than one keyword, it displays the results for each keyword separately and all possible combinations of those keywords. For example, searching in the webhelp format for environment variables (without quotes) gives results in three categories:

- Results for: environment

- Results for: variables

- Results for: environment,variables

Unlike the plain HTML format, the webhelp format does not support search terms within quotes.

Overview

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

Chapter 1: "About GT.M" (page 1) introduces GT.M administration and operations.

Chapter 2: "Installing GT.M" (page 4) provides procedures for installing GT.M.

Chapter 3: "Basic Operations" (page 13) describes operations required to start GT.M and keep it running.

Chapter 4: "Global Directory Editor" (page 45) 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 88) describes how to use a GT.M utility called MUPIP to perform database and non-database operations.

Chapter 6: "GT.M Journaling" (page 181) describes the journaling capabilities of GT.M.

Chapter 7: "Database Replication" (page 230) describes how to implement continuous application availability using multiple systems.

Chapter 8: "M Lock Utility (LKE)" (page 324) 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 338) provides an overview of GT.M Database Structure (GDS).

Chapter 10: "Database Structure Editor" (page 352) 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 400) describes procedures for verifying and maintaining the integrity of GDS databases.

Chapter 12: "Database Encryption" (page 441) describes procedures for encrypting data in the database and journal files.

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 13).

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 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 V6.3-012	08 April 2020	• In "GT.M Utility Programs" (page 2), Fixed GDE entry and added links to GDE and MUPIP chapters
Revision V6.3-001	20 March 2017	• In "Hardware/Operating System Environment" (page 1), remove references to OpenVMS • In "Security" (page 1), removed the reference to OpenVMS

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 variety of UNIX/Linux implementations. 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.

> **Note**
>
> The 32-bit version of GT.M is compiled to require the 586 ("Pentium compatible") instruction set.

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 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 45). 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. See Chapter 4: "Global Directory Editor" (page 45) for more information.

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. See Chapter 5: "General Database Management" (page 88) for more information.

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 324) 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 352) 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 hypen (-). 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 400), 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 [400] 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 484) for more information.

Chapter 2. Installing GT.M

Revision History		
Revision V7.0-001	24 November 2021	• In "Before you begin" (page 6), Installation no longer prompts for ICU version • In "gtminstall script " (page 10), gtminstall does not accept ICU version • In "Installation Procedure" (page 7), Installation does not prompt for ICU version; tar on AIX should be run as root
Revision V6.3-010	31 October 2019	• In "gtminstall script " (page 10), add the filename option.
Revision V6.3-007	04 February 2019	• In "Logging" (page 42), Downlodable dm_audit_listener.zip • In "Audit Logging facility" (page 41), add new section • In "Configuring the Restriction facility" (page 38), add Audit Principal Device section • In "Environment Variables" (page 19), improve the description of gtm_passwd and gtm_obfusaction_key.
Revision V6.3-006	26 October 2018	• In "Before you begin" (page 6), change "perform Unicode-related operations" to "perform UTF-8 mode operations". • In "Compiling ICU " (page 32), UTF-8 mode tweaks • In "Configuring and operating GT.M with Unicode® support (optional) " (page 31), UTF-8 mode tweaks • In "Environment Variables" (page 19), UTF-8 mode tweaks. • In "gtminstall script " (page 10), UTF-8 tweaks • In "Installation Procedure" (page 7), UTF-8 tweaks • In "Starting GT.M " (page 32), UTF-8 mode tweaks
Revision V6.3-005	03 July 2018	• In "Environment Variables" (page 19), Correct gtm_mstack_size default from 270 to 272 and add gtm_mstack_cirt. • In "ZSYSTEM and PIPE OPEN command restriction facility" (page 40), add new

		section for the ZSYSTEM and PIPE OPEN command restriction facility
Revision V6.3-004	23 March 2018	• In "Environment Variables" (page 19), add the description of the gtm_mstack_size • In "gtminstall script " (page 10), general improvements with emphasis on the use of the verbose option.
Revision V6.3-003	12 December 2017	• In "Configuring the Restriction facility" (page 38), update TRIGGER_MOD; add HALT and ZHALT
Revision V6.3-002	22 August 2017	• In "Configuring the Restriction facility" (page 38), add information about the Restriction facility • In "Environment Variables" (page 19), Update gtm_tpnotacidtime for WRITE /* and millisecond precision and add gtm_nontprestart_log_delta, gtm_nontprestart_log_first, and gtm_repl_instname. • In "gtminstall script " (page 10), add systemd/addRemoveIPC-no description and remove passive voice from the opening paragraph. • In "Installation Procedure" (page 7), add format for newer versions of ICU and update the script output for V6.3-002
Revision V6.3-001	20 March 2017	• In "GT.M Environment Setup " (page 15), gtm_dbkeys to gtmcrypt_config • In "Before you begin" (page 6), removed the reference to HP-UX, Solaris and Tru64 • In "Configuring and operating GT.M with Unicode® support (optional) " (page 31), added an ICU compatibility table to reflect GT.M's ability to use the default ICU on Linux, AIX 7 and AIX 6. Also, explained when to use gtm_icu_version. • In "Environment Variables" (page 19), added updates for V6.3-001 and corrected the alphabetical ordering. Add to the note on gtm_procstuckexec; added gtm_zstep, gtm_statsdir, gtm_statsshare,gtm_aio_nr_events • In "M mode and UTF-8 mode " (page 31), remove the reference to Solaris and HP-UX • In "Instructions" (page 10), improved the instructions for compiling the reference implementation plugin and removed parts about creating symlinks. • In "Compiling the Reference Implementation Plugin " (page 10), improved the

		instructions for compiling the reference implementation plugin and removed parts about creating symlinks.
Revision V6.2-002	17 June 2015	• In "Instructions" (page 10), added instructions for reviewing and editing the Makefile. • In "Environment Variables" (page 19), added the description of gtm_autorelink_ctlmax.
Revision V6.2-001	27 February 2015	• In "Installation Procedure" (page 7), improved the description of the use of /temporary/directory. • In "gtminstall script " (page 10), specified that gtminstall expects icu-config to be available to install UTF-8 mode support. • In "Starting GT.M " (page 32), added the step for setting the LC_CTYPE environment variables to the examples for starting GT.M with UTF-8 support. • Improved the description of "Configuring huge pages for GT.M x86[-64] on Linux" (page 35).
Revision V6.1-000/1	04 September 2014	Added a new section called "Compiling the Reference Implementation Plugin " (page 10).

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) 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).

• 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)**. 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 UTF-8 mode 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 UTF-8 mode. GT.M generates the distribution for UTF-8 mode only if ICU 3.6 (or above) is installed on your system. GT.M uses the system default 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.

> ⚠️ **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/ V6.3-002_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.

Example:

```
$ mkdir /tmp/unpackgtm ; cd /tmp/unpackgtm
```

or

```
$ mkdir /var/tmp/unpackgtm ; cd /var/tmp/unpackgtm
```

> ### 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 temporary 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; On AIX, if you use the default tar you have to unpack as root, or use GNU tar):

```
$ tar zxvf /Distrib/GT.M/gtm_V63002_linux_i586_pro.tar.gz -C /temporary/directory
```

> ### 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:

```
configure to install GT.M
gtm sets a default user environment and starts GT.M.
gtmbase 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 13).

To start the GT.M installation procedure, change to the temporary directory and execute the configure script:

```
#cd /temporary/directory
#./configure
```

The configure script displays a message like the following:

```
 GT.M Configuration Script
GT.M Configuration Script Copyright 2009, 2017 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 use of your GT.M distribution to the specified group. This is a security-related option. By default, GT.M does not restrict access by group.

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

Enter a directory name, such as */usr/lib/fis-gtm/V6.3-002_x86_64*. If the directory does not exist, the configure script displays a message like the following:

```
Directory /usr/lib/fis-gtm/V6.3-002_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*, UTF-8 functionality is not installed.

> **Note**
>
> GT.M requires at least ICU Version 3.6 to be installed. The installation script uses pkg-config (icu-config, if pkg-config is not available) to find system default ICU version.

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/V6.3-002_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.

Congratulations! GT.M is now installed on the system. Proceed to "Compiling the Reference Implementation Plugin " (page 10) section to identify and resolve shared library dependencies on your system and then Chapter 3: "Basic Operations" (page 13) to set up a default user environment.

Compiling the Reference Implementation Plugin

If you plan to use database encryption and TLS replication, you must compile the reference implementation plugin to match the shared library dependencies unique to your platform.

Instructions

Compile the reference implementation plugin as follows:

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

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

    The package names vary by distribution / version.

2. Unpack $gtm_dist/plugin/gtmcrypt/source.tar to a temporary directory, for example:

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

3. Follow the instructions in the README.

    *   Open Makefile with your editor; review and edit the common header (IFLAGS) and library paths (LIBFLAGS) in the Makefile to reflect those on your system.

    *   Define the gtm_dist environment variable to point to the absolute path for the directory where GT.M is installed

    *   Copy and paste the commands from the README to compile and install the encryption plugin with the permissions defined at install time

4. Compare the permissions of $gtm_dist/libgtmshr.so to the newly installed shared libraries in $gtm_dist/plugin. Adjust the permission of the newly installed libraries as necessary.

gtminstall script

gtminstall is a stand-alone GT.M installation facility that attempts to download the latest / current production GT.M distribution from sourceforge.net and install GT.M using reasonable defaults. Download the gtminstall script from http://sourceforge.net/projects/fis-gtm/files/GT.M Installer to a temporary directory. Change to that directory and provide execute permission to the gtminstall script.

```
$ cd /tmp
$ chmod +x gtminstall
```

If you expect to store and process Unicode® characters in the database, you should install GT.M with UTF-8 support. You should install GT.M with UTF-8 support even if you are not using Unicode® characters but anticipate to use them sometime in the future.

```
$ sudo ./gtminstall --verbose --utf8
```

> **Note**
>
> For UTF-8 support, the gtminstall script relies on pkg-config to get the system default ICU version. If pkg-config is not available/applicable, it uses icu-config.

If your application does not store and process Unicode® characters in the database, install GT.M in M mode only with the following command:

```
$ sudo ./gtminstall --verbose
```

--verbose displays diagnostic information while the script downloads GT.M from sourceforge.net and installs GT.M. The script aborts execution when it runs into an error. When this happens fix the error, and run the gtminstall script again.

gtminstall is also a part of the GT.M binary distribution. gtminstall 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 script debugging option 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
--filename filename		The name of the GT.M distribution tarball
--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); defaults to discard if not specified
--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 prompts 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		Install UTF-8 support using system default ICU version
--verbose -	*	Output diagnostic information as the script executes; default is to run quietly

Command line switches	*	Description

- 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.

If gtminstall finds that the environment is a Linux installation using systemd, it prompts the user for permission to insert addRemoveIPC=no into /etc/systemd/logind.conf and restart logind; if the user denies permission, gtminstall stops the installation after issuing instructions on how to perform the task independently and information on why it is necessary.

Example:

```
$ sudo ./gtminstall --verbose # installs latest version in M mode only
$ sudo ./gtminstall --utf8 --verbose # install latest version with UTF-8 mode support
$ sudo ./gtminstall --distrib /Distrib/GT.M V6.3-003A --verbose # install V6.3-003A from a local
▶ directory
```

▼

Chapter 3. Basic Operations

Revision History		
Revision V7.0-004	20 September 2022	• In "Audit Logging facility" (page 41), add information about AM_ENABLE • In "Configuring the Restriction facility" (page 38), Add LKE and LKECLEAR • In "Environment Variables" (page 19), gtm_db_create_ver affects MUPIP CREATE
Revision V7.0-003	24 June 2022	• In "Using huge pages" (page 36), LD_PRELOAD and where libhugtlbfs is now maintained
Revision V7.0-002	29 March 2022	• In "Environment Variables" (page 19), add gtm_malloc_limit ; gtm_autorelink_ctlmax - Add routine shared object integrity-check
Revision V7.0-001	24 November 2021	• In "Configuring the Restriction facility" (page 38), add ZSTEP to the ZBREAK restriction. • In "Environment Variables" (page 19), fix a minor typo and update the entry for gtm_non_blocked_write_retries
Revision V7.0-000	12 February 2021	• In "Environment Variables" (page 19), If gtm_tprestart_log_delta is defined and gtm_tprestart_log_first is not defined, GT.M initializes gtm_tprestart_log_first to 0
Revision V6.3-014	06 October 2020	• In "Environment Variables" (page 19), added gtm_coredump_filter, revised gtm_baktmpdir, gtm_snaptmpdir, remove references to the deprecated gtm_log, and specify that gtm_statshare is applied only after a db file is opened.
Revision V6.3-013	30 June 2020	• In "Environment Variables" (page 19), add more information about $gtm_tmp
Revision V6.3-012	08 April 2020	• In "Configuring the Restriction facility" (page 38), correct alphabetical order; add LIBRARY and LOGDENIALS. • In "Environment Variables" (page 19), deprecate gtm_fullblockwrites
Revision V6.3-011	20 December 2019	• In "Environment Variables" (page 19), add description of gtm_nofflf and gtm_hupenable • In "M mode and UTF-8 mode " (page 31), add information in ICU version configuration used by gtmprofile
Revision V6.3-010	31 October 2019	• In "Logging" (page 42), Added additional documentation to the Direct Mode Auditing (or APD) feature. Specifically, more

		information on how to compile and run the logger programs is added. • In "Environment Variables" (page 19), correct the definition of gtm_retention; add gtm_max_indrcache_count which got removed in revision V6.3-009 due to a documentation error.
Revision V6.3-009	27 June 2019	• In "Environment Variables" (page 19), add gtm_linktmpdir
Revision V6.3-008	24 April 2019	• In "Environment Variables" (page 19), add gtm_max_indrcache_count and gtm_max_indrcache_memory. • In "M mode and UTF-8 mode " (page 31), remove references to the Compiling ICU on GT.M Supported Platforms appendix.
Revision V6.2-001	27 February 2015	• In "Environment Variables" (page 19), corrected the description of gtm_tpnotacidtime and added the descriptions of gtm_boolean, gtm_autorelink_shm, and gtm_autorelink_keeptrn. • Updated "gtmprofile " (page 15) for V6.2-001 enhancements.
Revision V6.1-000	01 August 2014	In "Environment Variables" (page 19), improved the descriptions of gtm_tpnotacidtime and gtm_zquit_anyway and added the descriptions of gtmcrypt_config, gtmtls_passwd_<tlslabel>, and gtmcrypt_FIPS.
Revision V6.0-003/1	19 February 2014	In "Environment Variables" (page 19), improved the description of gtm_nocenable.
Revision V6.0-003	27 January 2014	• In "gtm " (page 18), added the -help and -? command line options. • In "Environment Variables" (page 19), added the description of gtm_ipv4_only and gtm_etrap.
Revision V6.0-001	27 February 2013	• Added the "Configuring huge pages for GT.M x86[-64] on Linux" (page 35) section. • In "Environment Variables" (page 19), added the description of gtm_side_effects, gtm_extract_nocol, and gtm_crypt_plugin environment variables.
Revision V6.0-000/1	21 November 2012	• In "Environment Variables" (page 19), added the description of the environment variables gtm_tprestart_log_delta, gtm_tprestart_log_first, gtm_tpnotacidtime, and gtm_zmaxptime. • Added a note which states that the environment variable gtm_log is ignored from V6.0-000 onwards.

Revision V6.0-000	19 October 2012	In "Environment Variables" (page 19), added the description of the environment variable gtm_custom_errors.

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. *gtmcshrc*: 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, gtmcshrc 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 gtmschrc.

4. *gtmbase*: detects the shell type and adds gtmprofile to .profile or gtmchsrc to .cshrc so the shell automatically sources gtmprofile or gtmschrc 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

On POSIX shells, gtmprofile helps you set an environment for single-user, non-replicated use of GT.M.

gtmprofile sets reasonable defaults for the following environment variables for normal GT.M operation:

```
gtmdir, gtm_dist, gtm_etrap, gtmgbldir, gtm_icu_version, gtm_log, gtm_principal_editing, gtm_prompt,
 ▸ gtm_repl_instance, gtm_retention, gtmroutines, gtm_tmp, gtmver
```

▼

You can set the following environment variables before sourcing gtmprofile or running the gtm script;

- *gtm_chset* - set this to "UTF-8" to run GT.M in UTF-8 mode; it defaults to M mode. As UTF-8 mode requires a UTF-8 locale to be set in LC_CTYPE or LC_ALL, if a locale is not specified, gtmprofile also attempts to set a UTF-8 locale. Since GT.M in UTF-8 mode often requires gtm_icu_version to be set, if it is not set, gtmprofile attempts to determine the ICU version on the system and set it. This requires the icu-config program to be installed and executable by gtmprofile.

- *gtmdir* - set this to define a directory for the environment set by gtmprofile; it defaults to $HOME/.fis-gtm.

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.
```

The $gtmroutines value set by the gtmprofile script enables auto-relink by default for object files in the $gtmdir/$gtmver/o directory in M mode and $gtmdir/$gtmver/o/utf8 in UTF-8 mode. Auto-relink requires shared memory resources and limits beyond those for database operation. If your system has inadequate shared memory configured, GT.M displays messages along the lines of:

```
%GTM-E-SYSCALL, Error received from system call shmget() failed
```

Refer to your OS documentation to configure shared memory limits (for example, on common Linux systems, the kernel.shmmax parameter in /etc/sysctl.conf).

The gtmprofile (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 if gtm_dist is already defined.

When gtm sources gtmprofile, it provides a default execution environment (global directory and a default database (with BEFORE_IMAGE journaling) if none exists. By default, it creates the database in $HOME/.fis-gtm with a structure like the following; note that this directory structure has different locations for GT.M routines (r), object files (o), and database-related files (g):

```
.fis-gtm
|-- r
|-- V6.2-000_x86_64
| |-- g
| | |-- gtm.dat
| | |-- gtm.gld
| | `-- gtm.mjl
| |-- o
| | `-- utf8
| `-- r
|-- V6.2-001_x86_64
 |-- g
 | |-- gtm.dat
 | |-- gtm.gld
 | `-- gtm.mjl
 |-- o
 | `-- utf8
 `-- r
```

where V6.2-001_x86_64 represents the current release and platform information and V6.2-000_x86_64 represents a previously used GT.M release.

On 64-bit platforms in M mode, gtmprofile sets the environment variable gtmroutines to something like the following (where $gtm_dist and $gtmver are as discussed above):

```
$gtmdir/$gtmver/o*($gtmdir/$gtmver/r $gtmdir/r) $gtm_dist/plugin/o($gtm_dist/plugin/r) $gtm_dist/libgtmutil.so
▶ $gtm_dist
```

🔻

$gtmdir/$gtmver/o*($gtmdir/$gtmver/r $gtmdir/r) specifies that GT.M searches for routines in $gtmdir/$gtmver/r, then $gtmdir/r, using $gtmdir/$gtmver/o for object code, then for routines in the plugin subdirectory of $gtm_dist, then in $gtm_dist, looking first for a shared library of routines distributed with GT.M and then for other routines subsequently installed there. The *-suffix after the object directory enables the auto-relink facility.

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 this:

```
$gtmdir/$gtmver/o/utf8*($gtmdir/$gtmver/r $gtmdir/r) $gtm_dist/plugin/o/utf8($gtm_dist/plugin/r)
▶ $gtm_dist/libgtmutil.so $gtm_dist
```

🔻

Note that gtmprofile sets $gtm_dist in UTF-8 mode to the utf8 subdirectory of the GT.M installation directory. If you have installed any plugins that include shared libraries, gtmprofile script includes those. For example, with the GT.M POSIX and ZLIB plugins installed on a 64-bit platform, gtmdir set to /home/jdoe1 and GT.M installed in /opt/fis-gtm/V6.2-001_x86_64, gtmprofile would set gtmroutines to:

```
/home/jdoe1/.fis-gtm/V6.2-001_x86_64/o*(/home/jdoe1/.fis-gtm/V6.2-001_x86_64/r /home/jdoe1/.fis-gtm/r)
▶ /usr/lib/fis-gtm/V6.2-001_x86_64/plugin/o/_POSIX.so /usr/lib/fis-gtm/V6.2-001_x86_64/plugin/o/_ZLIB.so /usr/
lib/fis-gtm/V6.2-001_x86_64/plugin/o(/usr/lib/fis-gtm/V6.2-001_x86_64/plugin/r) /usr/lib/fis-gtm/V6.2-001_x86_64/
libgtmutil.so /usr/lib/fis-gtm/V6.2-001_x86_64
```

🔻

> **Note**
>
> This scenario of sourcing gtmprofile is only for the sake of example. Consult your system administrator before implementing gtmprofile for a multi-user environment.

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.

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 run can 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 LC_CTYPE to a UTF-8 locale and then run the gtm script.

If you intend to use Database Encryption, set the gtm_passwd and gtmcrypt_config environment variables 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 are as 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).

GTMCI specifies the call-in table for function calls from C code to M code.

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 (LIBPATH on AIX) is a standard system environment variable used to modify the default library search path. Use this extension when GT.M relies on custom compiled libraries that do not reside in the default library search path. For example ICU, GPG, OpenSSL and/or zlib libraries.

TZ is a standard system environment variable that specifies the timezone to be used by a GT.M process, if they are not to use the default system timezone. GT.M uses the system clock for journal time stamps on the assumption it reflects UTC time.

gtmcompile specifies the initial value of the $ZCOmpile ISV. The SET command can alter the value of $ZCOMPILE in an active process.

gtmcrypt_config specifies the location of the configuration file required for database encryption, Sequential file, PIPE, and FIFO device encryption and/or TLS support. A configuration file is divided into two sections: database and TLS. The "database" section contains a list of database files and their corresponding key files. You do not need to add a database section if you are not using an encrypted database, or a "tls" section if you are not using TLS for replication or sockets. The tls section provides information needed for OpenSSL (in the reference plugin implementation) or other encryption package, such as the location of root certification authority certificate in PEM format and leaf-level certificates with their corresponding private key files. Note that the use of the gtmcrypt_config environment variable requires prior installation of the libconfig package.

gtmcrypt_FIPS specifies whether the plugin reference implementation should attempt to use either OpenSSL or Libgcrypt to provide database encryption that complies with FIPS 140-2. When the environment variable $gtmcrypt_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 libgcrypt (from GnuPG) and libcrypto (OpenSSL) in "FIPS mode." Note that to comply with FIPS 140-2 you should be knowledgeable with that standard and take many steps beyond setting this environment variable. By default GT.M does not enforce "FIPS mode."

gtmgbldir 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 use this environment variable to conveniently choose one to use from the time of process startup. To automate this definition, define gtmgbldir in the user's login file. The SET command can alter the value of $ZGBLDIR in an active process.

gtmroutines specifies the initial value of the $ZROutines ISV, which specifies where to find object and source code. The SET command can alter the value of $ZROUTINES in an active process.

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.

gtm_aio_nr_events: For Linux x86_64, the gtm_aio_nr_events environment variable controls the number of structures a process has per global directory to manage asynchronous writes, and therefore determines the number of concurrent writes a process can manage across all regions within a global directory. If not specified, the value controlled by gtm_aio_nr_events defaults to 128. If a process encounters a situation where it needs to perform an asynchronous write, but has no available slots with which to manage an additional one, it either falls back to synchronous writing if the write is blocking other actions, and otherwise defers the write until a slot becomes available as other writes complete. Linux allocates the structures on a system-wide basis with the setting of /proc/sys/fs/aio-max-nr. Therefore you should configure this parameter to account for the needs (as determined by gtm_aio_nr_events or the default) of all processes using asynchronous I/O. When processes use multiple global directories with asynchronous I/O, their need for the system resources increases accordingly. For example, if an environment runs 10,000 processes each of which open two global directories and /proc/sys/fs/aio-max-nr is set to a value of 200,000 then gtm_aio_nr_events needs to be set to a value <= 200,000 / (10,000 * 2) = 10. Conversely if gtm_aio_nr_events is set to a value of 20, then aio-max-nr needs to be bumped up to (10,000 * 2 * 20) = 400,000. GT.M captures the number of errors encountered when attempting to write database blocks for a region, and, barring problems with the storage subsystem, hitting an asynchronous write limit would constitute primary (probably only) contribution to that value, which you can access with $$^%PEEKBYNAME("sgmnt_data.wcs_wterror_invoked_cntr",<region>)

gtm_autorelink_ctlmax specifies the maximum number of entries for unique routine names in the relink control file created by a process for any directory, with a minimum of 1,000, a maximum of 16,000,000 and a default of 50,000 if unspecified. If a specified value is above or below the allowed range, the process logs the errors ARCTLMAXHIGH or ARCTLMAXLOW respectively in the syslog, and uses the nearest acceptable limit instead. MUPIP RCTLDUMP and ZSHOW "A" outputs include the maximum number of unique routine names available in a relink control file. If the maximum number of entries for unique routine names is exceeded, it triggers a relinkctl integrity-check to ensure that all pointers and indexes are within the shared memory bounds.

gtm_autorelink_keeprtn:

When gtm_autorelink_keeprtn is set to 1, t[rue], or y[es] , exiting processes leave auto-relinked routines in shared memory. When the environment variable gtm_autorelink_keeprtn is undefined, 0, f[alse] or n[o], exiting processes purge auto-relinked routines in shared memory if no other processes are using them. Regardless of the value of gtm_autorelink_keeprtn, the Operating System removes an auto-relink shared memory repository when there are no processes accessing it.

All values are case-independent. When gtm_autorelink_keeprtn is defined and TRUE:

- Process do less work on exiting, with some performance gain - faster process termination - likely only observable when a large number of processes exit concurrently.

- In a production environment, an application that frequently invokes GT.M routines in short running processes (such as GT.M routines invoked by web servers using interfaces such as CGI) may give better performance when setting gtm_autorelink_keeprtn or using at least one long running auto-relink process that remains attached to the shared memory to keep routines available in shared memory for use when short running processes need them.

gtm_autorelink_shm specifies the size (in MiB) of an initial Rtnobj shared memory segment used by the auto-relink facility. If the value of gtm_autorelink_shm is not a power of two, GT.M rounds the value to the next higher integer power of two. If the first object (.o) file does not fit in a new Rtnobj segment, GT.M rounds the allocation up to the smallest integer power of two required to make it fit. When GT.M needs room for object files, and existing Rtnobj segments have insufficient free space, it creates an additional shared memory segment, double the size of the last. Note that when hugepages are enabled, the actual Rtnobj shared memory size might be more than that requested implicitly or explicitly through $gtm_autorelink_shm.

gtm_badchar specifies the initial setting that determines whether GT.M should raise an error when it encounters an illegal UTF-8 character sequence. This setting can be changed with a VIEW "[NO]BADCHAR" command, and is ignored for I/O processing and in M mode.

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 both $gtm_baktmpdir and $GTM_BAKTMPDIR are not defined, MUPIP BACKUP creates temporary files as follows:

- MUPIP BACKUP -DATABASE uses the directory of the backup destination for creating temporary files.

- MUPIP BACKUP -BYTESTREAM uses /tmp.

All processes performing updates during an online BACKUP must have the use the same directory and have write access.

gtm_boolean specifies the initial setting that determines how GT.M compiles Boolean expression evaluation (expressions evaluated as a logical TRUE or FALSE). If gtm_boolean is undefined or evaluates to an integer zero (0), GT.M behaves as it would after a VIEW "NOFULL_BOOLEAN" and compiles such that it stops evaluating a Boolean expression as soon as it establishes a definitive result . Note that:

- $gtm_side_effects has an analogous impact on function argument evaluation order and implies "FULLBOOLEAN" compilation, so VIEW "NOFULLBOOLEAN" produces an error when $gtm_side_effects is on.

- If gtm_boolean evaluates to an integer one (1), GT.M enables VIEW "FULL_BOOLEAN" compilation, which means that GT.M ensures that, within a Boolean expression, all side effect expression atoms, extrinsic functions ($$), external functions ($&), and $INCREMENT() execute in left-to-right order.

- If gtm_boolean evaluates to an integer two (2), GT.M enables VIEW "FULL_BOOLWARN" behavior, which means that GT.M not only evaluates Boolean expressions like "FULL_BOOLEAN" but produces a BOOLSIDEFFECT warning when it encounters Boolean expressions that may induce side-effects; that is: expressions with side effects after the first Boolean operator - extrinsic functions, external calls, and $INCREMENT().

gtm_chset determines the mode in which GT.M compiles and 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" (or 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 using non-M standard (ASCII) collation. 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_coredump_filter: In Linux x86_64 if the environment variable gtm_coredump_filter is defined, it specifies the mappings of the process address space for a GT.M process, with the bits having the same meaning as those specified for /proc/{pid}/coredump_filter in "man 5 core". If unspecified, GT.M uses a value of 0x73; a value of -1 prevents GT.M from modifying the coredump_filter value. A running process can change its coredump_filter by writing to the file /proc/{pid}/coredump_filter, and can query the current value by reading that file.

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/libgtmcrypt.so to be a symbolic link to a shared library providing the plugin. The expected name of the actual shared library is libgtmcrypt_cryptlib_CIPHER.so (depending on your platform, the actual extension may differ from .so), for example, libgtmcrypt_openssl_AESCFB. GT.M cannot and does not ensure that the cipher is actually AES CFB as implemented by OpenSSL.

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_db_create_ver directs MUPIP CREATE to create GT.M V6.3-014 compatible database file(s) when the environment variable is set to "6" or "V6".

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 extremely 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_procstuckexec mechanism at approximately 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_procstuckexec 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_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 GT.M system distribution for Unicode® support. The distribution for UTF-8 mode 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_dmterm specifies a [NO]DMTERM state at process initiation where application setting applied to $PRINCIPAL also apply to direct mode interactions; a case-insensitive value of "1", "yes", or "true" establishes a DMTERM state at process initiation where direct mode uses default terminal characteristics and ignores application settings for $PRINCIPAL; all other values, including no value, result in the default VIEW "NODMTERM" behavior.

gtm_env_translate specifies the path to a shared library to implement the optional GT.M environment translation facility that can assist in resolving extended global referernces.

gtm_etrap specifies an initial value of $ETRAP to override the default value of "B" for $ZTRAP as the base level error handler. The gtmprofile script sets gtm_etrap to **"Write:(0=$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 used by application logic.

gtm_nofflf specifies the default WRITE # behavior for STREAM and VARIABLE format sequential files. If it is set to 1, Y[ES] or T[RUE], WRITE # writes only a form-feed <FF> character in conformance to the M standard. If it is not defined or set to 0, NO, or FALSE, WRITE # writes <FF><LF> characters. The [NO]FFLF deviceparameter for USE and OPEN commands takes precedence over any value of gtm_nofflf.

gtm_gdscert specifies the initial setting that controls whether GT.M processes should test updated database blocks for structural damage. If it is defined, and evaluates to a non-zero integer or any case-independent string or leading substrings of "TRUE" or "YES", GT.M performs block-level integrity check on every block as a process commits it. Within a running process, VIEW "GDSCERT":value controls this setting 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 that determines 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", when 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. Within a running process, VIEW "GVDUPSETNOOP":value controls this behavior. By default, GT.M enables this optimization.

gtm_hupenable specifies the initial value that determines whether a GT.M process should recognize a disconnect signal from a PRINCIPAL device that is a terminal. If it is defined, and evaluates to a non-zero integer or any case-independent string or leading substrings of "TRUE" or "YES", the proces receives an TERMHANGUP error if the O/S signals that the terminal assigned to the process as the PRINCIPAL device has disconnected. Within a running process, USE $PRINCIPAL: [NO]HUP[ENABLE] controls this behavior. By default, GT.M ignores such a signal, but a process that ignores the signal may subsequently receive an IOEOF or a TERMWRITE error from an attempt to respectively READ from, or WRITE to the missing device. GT.M terminates a process that ignores more than one of these messages and, if the process is not in Direct Mode, sends a NOPRINCIO message to the operator log.

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. Please see the section on "Configuring and operation GT.M with Unicode® Support" for more information,

gtm_ipv4_only specifies whether a Source Server should establish only IPv4 connections with a Receiver Server or sockets associated with a SOCKET device. 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. gtm_ipv4_only is useful 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 leaves object files; by default, gtminstall deletes object files after archiving them in a shared library.

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_link specifies the initial setting that determines whether GT.M permits multiple versions of the same routine to be active at different stack levels of the M virtual machine. The VIEW "LINK":"[NO]RECURSIVE" command modifies this in an active process. If gtm_link is set to "RECURSIVE", auto-relink and explicit ZLINK commands links a newer object even when a routine with the same name is active and available in the current stack. When a process links a routine with the same name as an existing routine, future calls use the new routine. Prior versions of that routine referenced by the stack remain tied to the stack until they QUIT, at which point they become inaccessible. This provides a mechanism to patch long-running processes. If gtm_link is undefined or set to NORECURSIVE, or any value other than "RECURSIVE", auto-zlink defers replacing older routines until they no longer have an invoking use by the process and a ZLINK command produces a LOADRUNNING error when it attempts to relink an active routine on the GT.M invocation stack.

gtm_local_collate specifies an alternative collation sequence for local variables.

gtm_linktmpdir provides a location into GT.M creates small temporary files used for auto-relink; if unspecified, this location defaults to $gtm_tmp, which in turn defaults to /tmp. The implications for group and permissions of the directory are described in the section Shared Resources Authorization Permissions of Appendix D: "GT.M Security Philosophy" (page 496). FIS recommends that all processes that share a directory whose contents are subject to ZRUPDATE use the same value for $gtm_linktmpdir so that all processes see update notifications - with different values of $gtm_linktmpdir, a ZRUPDATE by a process with one value of $gtm_linktmpdir would not be observed by a process with a different value of that environment variable.

gtm_max_indrcache_count (use only under the guidance of your GT.M support channel) specifies the number of cached entries in indirection compilation. The default is 128.

gtm_malloc_limit specifies an initial value for $ZMALLOCLIM. The default is 0, which indicates no warning limit on process memory utilization. When the environment variable specifies a positive value, GT.M uses the smaller of that value and any OS defined amount for the value of $ZMALLOCLIM. GT.M does not give errors or messages about its choice for $ZMALLOCLIM between a specified value and some other more appropriate value, so if the application needs to verify the result, it should examine the resulting ISV value For additional information, refer to the $ZMALLOCLIM section in the Intinsic Special Variable chapter of the **GT.M Programmer's Guide**.

gtm_max_indrcache_memory (use only under the guidance of your GT.M support channel) specifies the maximum size (in KiB) of memory that GT.M should use for indirection cache object code. The default is 128.

gtm_max_sockets specifies the maximum number of client connections for socket devices. The default is 64.While it must be large enough to accommodate the actual need, each reservation requires some memory in socket structures, so setting this number unnecessarily high causes requires a bit of additional memory for no benefit.

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 (KiB). Setting this too low can impede investigations of memory issues, but GT.M only uses this reserve when a process runs out of memory so it almost never requires actual memory, only address space.

gtm_mstack_crit specifies an integer between 15 and 95 defining the percentage of the stack which should be used before GT.M emits a STACKCRIT warning. If the value is below the minimum or above the maximum GT.M uses the minimum or maximum respectively. The default is 90.

gtm_mstack_size specifies the M stack size (in KiB). If gtm_mstack_size is not set or set to 0, GT.M uses the default M stack size (that is, 272KiB). The minimum supported size is 25 KiB; GT.M reverts values smaller than this to 25 KiB. The maximum supported size is 10000 KiB; GT.M reverts values larger than this to 10000 KiB.

gtm_mupjnl_parallel defines the number of processes or threads used by MUPIP JOURNAL -RECOVER/-ROLLBACK when the invoking command does not have a -PARALLEL qualifier. When defined with no value, it specifies one process or thread per region. When undefined or defined to one (1), it specifies MUPIP should process all regions without using additional processes or threads. When defined with an integer value greater than one (1), it specifies the maximum number of processes or threads for MUPIP to use. If the value is greater than the number of regions, MUPIP never uses more processes or threads than there are regions. If it is less than the number of regions, MUPIP allocates work to the additional processes or threads based on the time stamps in the journal files.

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 the write behavior of FIFO, PIPE, or non-blocking sockets. GT.M retries a WRITE which would block up to $gtm_non_blocked_retries times with a 100 milliseconds delay between each retry. The default value is 10 times.

gtm_nontprestart_log_delta specifies the number of non-transaction restarts for which GT.M should wait before reporting a non-transaction restart to the operator logging facility. If gtm_nontprestart_log_delta is not defined, GT.M initializes gtm_nontprestart_log_delta to 0.

gtm_nontprestart_log_first specifies the initial number of non-transaction restarts which GT.M should report before placing non-transaction restart reports to the operator logging facility using the gtm_nontprestart_log_delta value. If gtm_nontprestart_log_delta is defined and gtm_nontprestart_log_first is not defined, GT.M initializes gtm_nontprestart_log_first to 1.

gtm_noundef specifies the initial setting that controls whether a GT.M process should treat undefined global or local 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. The VIEW "[NO]UNDEF" command can alter this behavior in an active process. By default, GT.M signals an error on an attempt to use the value of an undefined variable.

gtm_obfuscation_key: If $gtm_obfuscation_key specifies the name of file readable by the process, the encryption reference plugin uses a cryptographic hash of the file's contents as the mask for the obfuscated password in the environment variable gtm_passwd. When gtm_obfuscation_key does not point to a readable file, the plugin computes a cryptographic hash using a mask based on the value of $USER and the inode of the mumps executable to use as a mask. $gtm_passwd set with a $gtm_obfuscation_key allows access to all users who have the same $gtm_obfuscation_key defined in their environments. However, $gtm_passwd set without $gtm_obfuscation_key can be used only by the same $USER using the same GT.M distribution.

gtm_passwd specifies the obfuscated (not encrypted) password of the GNU Privacy Guard key ring. When the environment variable $gtm_passwd is set to "", GT.M invokes the default GTMCRYPT passphrase prompt to obtain a passphrase at process startup and uses that value as $gtm_passwd for the duration of the process.

gtm_patnumeric specifies the value of the read-only ISV $ZPATNUMERIC that determines how GT.M interprets the patcode "N" used in the pattern match operator. The SET command can alter the value of $ZPATNUMERIC in an active process. If the environment variable gtm_patnumeric is not defined or set to a value other than "UTF-8", GT.M initializes $ZPATNUMERIC to "M".

gtm_pattern_file and **gtm_pattern_table** specify alternative patterns for the pattern (?) syntax. Refer to the Internationalization chapter in the Programmer's Guide for additional information.

gtm_poollimit restricts the number of global buffers a process uses in order to limit the potential impact on other processes. It is intended for use by a relatively small subset of processes when those processes have the potential to "churn" global buffers; the value is of the form n[%] when it ends with a per-cent sign (%), the number is taken as an as a percentage of the configured global buffers and otherwise as an ordinal number of preferred buffers; standard M parsing and integer conversions apply. Note that this environment variable applies to all regions accessed by a process; the VIEW command for this feature allows finer grained control. MUPIP REORG uses this facility to limit its buffers with a default of 64 if gtm_poollimit is not specified. Note that this may slightly slow a standalone REORG but can be overridden by defining gtm_poollimit as 0 or "100%".

gtm_principal specifies the value for $PRINCIPAL, which designates an alternative name (synonym) for the principal $IO device.

gtm_principal_editing specifies the initial settings for $PRINCIPAL of the following colon-delimited deviceparameters: [NO]EDITING [NO]EMPTERM and [NO]INSERT; in an active process the USE command can modify these device characteristics.

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

gtm_procstuckexec specifies a shell command or a script to execute when any of the following conditions occur:

- A one minute wait on a region due to an explicit MUPIP FREEZE or an implicit freeze, such as BACKUP, INTEG -ONLINE, and so on.

- 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.

- A process encounters conditions that produce the following operator log messages: BUFOWNERSTUCK, INTERLOCK_FAIL, JNLPROCSTUCK, SHUTDOWN, WRITERSTUCK, MAXJNLQIOLOCKWAIT, MUTEXLCKALERT, SEMWT2LONG, and COMMITWAITPID.

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. For example for the script to invoke the debugger, the process must be of the same group or have a way to elevate priveleges.

gtm_prompt specifies the initial value of the ISV $ZPROMPT, which controls the GT.M direct mode prompt. The SET command can alter the value of $ZPROMPT in an active process.By default, the direct mode prompt is "GTM>".

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_instname specifies a replication instance name that uniquely identifies an instance. The replication instance name is immutable. The maximum length of a replication instance name is 15 bytes. Note that the instance name is not the same as the name of the replication instance file (gtm_repl_instance). You need to specify a replication instance name at the time of creating an replication instance file. If you do not define gtm_repl_instname, you need to specify an instance name using -NAME=<instance_name> with MUPIP REPLICATE -INSTANCE_CREATE.

gtm_repl_instsecondary specifies the name of the replicating instance in the current environment. GT.M uses $gtm_repl_instsecondary if the -instsecondary qualifer is not specified.

gtm_retention (used by the gtm/gtmprofile script) - On every invocation, the gtm script deletes all prior generation journal files (*_<time and date stamp>) older than $gtm_retention days from the directory (and its subdirectories) of the global directory. By default, $gtm_retention is 42.

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 actuallist 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 SIDEEFFECTEVAL 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 SIDEEFFECTEVAL 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 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). Note that warning reports for the two features are separately controlled by setting their values to 2. The differences in the compilation modes may include not only differences in results, but differences in flow of control when the code relies on side effect behavior.

gtm_snaptmpdir specifies the location to place the temporary "snapshot" file created by facilities such as MUPIP INTEG -ONLINE. If $gtm_snaptmpdir is not defined, GT.M uses the deprecated $GTM_BAKTMPDIR environment variable. If $GTM_BAKTMPDIR is not defined, MUPIP INTEG uses the current working directory. All processes performing updates during an MUPIP INTEG -ONLINE use the same snapshot directory and have write access.

gtm_statsdir specifies the directory for database files into which processes opted-in to sharing global statistics place their statistics as binary data. If you do not explicitly define this environment variable for a process, GT.M defines this to the evaluation of $gtm_tmp, which defaults to /tmp. All processes that share statistics MUST use the same value for $gtm_statsdir. FIS suggests that you point gtm_statsdir at a tmpfs or ramfs on Linux, and a filesystem in a ram disk on AIX. These database files have a name derived from the user defined database file name and a .gst extension. They are not usable as normal database files by application code, except to read statistics. GT.M automatically creates and deletes these database files as needed. Under normal operation, applications do not need to manage them explicitly. The mapping of ^%YGS to statistics database files is managed by GT.M within global directories, transparently to applications. The ^%YGBLSTAT utility program gathers and reports statistics from nodes of ^%YGS(region,pid).

gtm_statshare specifies an initial value for the characteristic controlled by VIEW "[NO]STATSHARE" in application code. A value of 1, or any case-independent string or leading substrings of "TRUE" or "YES" in the environment variable gtm_statshare provides the equivalent of VIEW "STATSHARE" as the initial value when a database file is opened. Leaving the gtm_statshare undefined or defined to another value, typically 0, "FALSE" or "NO" provides the equivalent of VIEW "NOSTATSHARE" as the initial value.

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

gtm_string_pool_limit when positive, specifies the initial value in bytes for $ZSTRPLLIM.

gtm_sysid specifies the value for the second piece of the $SYstem ISV. $SYSTEM contains a string that identifies the executing M in stance. 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 GT.M stores socket files for communication between gtmsecshr and GT.M processes. FIS recommends setting the environment variable gtm_tmp to a location:

- which is unique to a GT.M version. For example, /temp/V6.3-014_x86_64

- where operation procedures and policies can ensure that the contents are not removed when there are active GT.M processes.

If appropriate, set gtm_tmp to a directory location in tmpfs or ramfs on Linux or a filesystem in a ram disk on AIX.

If gtm_tmp is not defined, GT.M uses the /tmp directory which may disrupt active gtmsecshr operations when /tmp is either cleared manually or by the retention policies of the operating system.

gtm_tpnotacidtime specifies the maximum time that a GT.M process waits for non-Isolated timed command (HANG, JOB, LOCK, OPEN, READ, WRITE /* 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>2). gtm_tpnotacidtime specifies time in seconds to millisecond precision (three decimal places); the default is 2 seconds. The maximum value of gtm_tpnotacidtime is 30 and the minimum is 0. If gtm_tpnotacidtime 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_tpnotacidtime). 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_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 which GT.M should report before pacing transaction restart reports to the operator logging facility using the gtm_tprestart_log_delta value. If gtm_tprestart_log_delta is defined and gtm_tprestart_log_first is not defined, GT.M initializes gtm_tprestart_log_first to 0.

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 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 processes that otherwise don't open a database file (such as GDE) can cause them to encounter an error.

gtm_trigger_etrap 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. The SET command can alter the value of $ZDATE in an active process.

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. The SET command can alter the value of $ZINTERRUPT in an active process.

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_zmaxptime 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_zmaxptime specifies time in seconds and the default is 0, which indicates "no timeout" (unlimited time). The maximum value of gtm_zmaxptime is 60 seconds and the minimum is 0; GT.M ignores gtm_zmaxptime if it contains a value outside of this recognized range. This range check does not apply to SET $ZMAXTPTIME.

gtm_zquit_anyway specifies whether the code of the form QUIT <expr> execute as if it were SET <tmp>=<expr> QUIT:$QUIT tmp QUIT, where <tmp> is a temporary local variable in the GT.M runtime system that is not visible to application code. This setting is a run-time setting, rather than a compiler-time setting. If gtm_zquit_anyway is defined and evaluates to 1 or any case-independent string or leading substrings of "TRUE" or "YES", code of the form QUIT <expr> executes as if it were SET <tmp>=<expr> QUIT:$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", GT.M executes QUIT <expr> as specified by the standard..

gtm_zstep specifies the initial value of $ZSTEP, which defines the ZSTEP action; if gtm_zstep is not defined, $ZSTEP defaults to "B".

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 $ZTRAP also implicitly performs a NEW $ZTRAP before the SET.

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_passw (not used by GT.M directly) - It is used by the gtmprofile script in maintaining gtm_passwd.

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
LC_CTYPE	gtmprofile
gtmgbldir*	gtmprofile, gtmcshrc
gtmroutines*	gtmprofile, gtmcshrc
gtmver	gtmprofile
gtm_dist*	gtmprofile, gtmschrc
gtm_icu_version	gtmprofile
gtm_principal_editing	gtmprofile
gtm_prompt	gtmprofile
gtm_repl_instance	gtmprofile
gtm_retention	gtmprofile
gtm_tmp	gtmprofile
old_gtmroutines	gtmprofile
old_gtm_dist	gtmprofile
old_gtmver	gtmprofile
tmp_gtm_tmp	gtmpropfile
* denotes environment variables that must be defined for normal GT.M operation.	

Environment Variables	Set up by GT.M shell scripts
tmp_passw	gtmprofile
* 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 **$gtm_dist**). If multiple processes are accessing the same GT.M version, each process must use the same **$gtm_tmp**.

2. The **gtmprofile** script sets **$gtm_tmp** to **/tmp/fis-gtm/$gtmver.** On production systems, set $gtm_tmp as described earlier in this section.

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® support 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 support 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 libgtmutil.so therein) rather than its parent directory.

In addition to $gtm_chset, GT.M V5.3-004 and up use $gtm_icu_version to choose an ICU library library version other than the default. For ICU libraries built with **symbol renaming enabled**, $gtm_icu_version becomes a required setting.

$gtm_icu_version specifies the ICU version that GT.M should use for UTF-8 operations. It is in the form of **MajorVersion.MinorVersion** where MajorVersion and MinorVersion specify the desired Major verion and Minor version of ICU. For example, 3.6 refers to ICU version 3.6. If $gtm_icu_version is defined, GT.M works regardless of whether or not

symbols are renamed in ICU. If $gtm_icu_version is not defined or does not evaluate to an installed ICU version, GT.M look for non-renamed symbols in the default ICU version. Note that display widths for a few characters are different starting in ICU 4.0.

GT.M versions V5.2-000 to V5.3-003 required ICU 3.6 libraries. These versions did not use $gtm_icu_version. The following table lists GT.M versions and their compatibility with the operating system provided ICU libraries.

Operating System	Supported GT.M Versions	Compatibility with OS provided ICU libraries	Defin $gtm
AIX 7	GT.M V5.4-001 and up	GT.M can use the AIX provided ICU libraries which are compiled without symbol renaming.	N
AIX 6	GT.M V5.3-004 until GT.M V6.2-002A	GT.M cannot use the AIX provided ICU libraries. Please build and install an appropriate ICU version and define the gtm_icu_version environment variable accordingly.	Y
	GT.M V6.3-000 and up	GT.M can use the AIX provided ICU libraries which are compiled without symbol renaming.	N
Linux (RHEL 6+, SLES 11+, Ubuntu 12.04+)	GT.M V5.3-004 to GT.M V5.4-001	GT.M can use the Linux distribution provided ICU libraries from ICU 3.6 until ICU 4.2. These libraries are compiled with symbol renaming.	Y
	GT.M V5.4-002 and up	GT.M can use the Linux distribution provided ICU libraries. These libraries are compiled with symbol renaming.	Y

> **Note**
>
> The **gtmprofile** script defines **$gtm_icu_version** as necessary.

Compiling ICU

GT.M uses ICU 3.6 (or above) to perform operations related to Unicode® support. GT.M generates the distribution for UTF-8 mode 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 support.

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. Please see the above section for more information.

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* and LC_CTYPE or LC_ALL to any usable UTF-8 locale.

```
$ export gtm_chset="UTF-8"
$ export LC_CTYPE="en_US.utf8"
```

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>/gtmcshrc
```

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 and LC_CTYPE or LC_ALL to any usable UTF-8 locale.

```
$ setenv gtm_chset UTF-8
$ setenv LC_CTYPE en_US.utf8
```

2. Source the gtmchrc script to set up default GT.M UTF-8 environment.

```
$ source <path_to_gtm_installation_directory>/gtmcshrc
```

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.

3. Ensure that you have set an appropriate value for TERM.

4. Consider adding 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 utlities.

```
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 uft8 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® support related GT.M functionality.

3. Ensure that you have installed ICU 3.6 (or above) and have LC_CTYPE or LC_ALL set to a usable UTF-8 locale.

4. Set LD_LIBRARY_PATH and TERM to appropriate values.

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

6. Add gtm_dist to the system environment variable PATH.

7. Set gtm_chset to UTF-8.

8. Consider adding these environment variables in your login file so you do not have to create them again the next time you start your shell.

9. 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"
```

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

```
$ gtm
```

11. 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 shell globbing to resolve wild-cards (* and ?) in 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 your shell, 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> do ^TEST** links the object file TEST.o and executes it; if the **TESTM** program calls other M routines, those are automatically compiled and linked.

> ⚠️ **Caution**
>
> When possible, GT.M verifies that MUMPS, MUPIP, DSE and LKE reside in $gtm_dist. If the path to the executable and the path to $gtm_dist do not match each executable issues an error. In cases where the executable path could not be determined, each executable defers issuing an error until it is required.

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

Huge pages are a Linux feature that may improve the performance of GT.M applications in production. Huge pages create a single page table entry for a large block (typically 2MiB) of memory in place of hundreds entries for many smaller (typically 4KiB) blocks. This reduction of memory used for page tables frees memory for other uses, such as file system caches, and increases the probability of TLB (translation lookaside buffer) matches, both of which can improve performance. The performance improvement related to reducing the page table size becomes evident when many processes share memory as they do for global buffers, journal buffers, and replication journal pools. Configuring huge pages on Linux for x86 or x86_64 CPU architectures help improve:

1. **GT.M shared memory performance:** When your GT.M database uses journaling, replication, and the BG access method.

2. **GT.M process memory performance:** For your process working space and dynamically linked code.

> 📌 **Note**
>
> At this time, huge pages have no effect for MM databases; the text, data, or bss segments for each process; or for process stack.

While FIS recommends you configure huge page for shared memory, you need to evaluate whether or not configuring huge page for process-private memory is appropriate for your application. Having insufficient huge pages available during certain

commands (for example, a JOB command - see complete list below) can result in a process terminating with a SIGBUS error. This is a current limitation of Linux. Before you use huge pages for process private memory on production systems, FIS recommends that you perform appropriate peak load tests on your application and ensure that you have an adequate number of huge pages configured for your peak workloads or that your application is configured to perform robustly when processes terminate with SIGBUS errors. The following GT.M features fork processes and may generate SIGBUS errors when huge pages are not available-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, and replication-related MUPIP commands that start server processes and/or helper processes. As increasing the available huge pages 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.

Consider the following example of a memory map report of a Source Server process running at peak load:

```
$ pmap -d 18839
18839: /usr/lib/fis-gtm/V6.2-000_x86_64/mupip replicate -source -start -buffsize=1048576 -secondary=melbourne:1235
▶ -log=/var/log/.fis-gtm/mal2mel.log -instsecondary=melbourne
Address   Kbytes Mode Offset   Device Mapping
--- lines removed for brevity -----
mapped: 61604K writeable/private: 3592K shared: 33532K
$
```

▼

Process id 18839 uses a large amount of shared memory (33535K) and can benefit from configuring huge pages for shared memory. Configuring huge pages for shared memory does not cause a SIGBUS error when a process does a fork. For information on configuring huge pages for shared memory, refer to "Using huge pages" and "Using huge pages for shared memory" sections. SIGBUS errors only occur when you configure huge pages for process private memory; these errors indicate you have not configured your system with an adequate number of huge pages. To prevent SIGBUS errors, you should perform peak load tests on your application to determine the number of required huge pages. For information on configuring huge pages for process private memory, refer to "Using huge pages" and "Using huge pages for process working space" sections.

As application response time can be deleteriously affected if processes and database shared memory segments are paged out, FIS recommends configuring systems for use in production with sufficient RAM so as to not require swap space or a swap file. While you must configure an adequate number of huge pages for your application needs as empirically determined by benchmarking / testing, and there is little downside to a generous configuration to ensure a buffer of huge pages available for workload spikes, an excessive allocation of huge pages may affect system throughput by reserving memory for huge pages that could otherwise be used by applications that cannot use huge pages.

Using huge pages

Prerequisites	Notes
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**
libhugetlbfs.so	Use your Linux system's package manager to install the libhugetlbfs.so library 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.
Have 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

Prerequisites	Notes
	memory is still available, with the command: **hugeadm --pool-pages-min DEFAULT:num_pages**
	For subsequent on-demand allocation of Huge Pages, use: **hugeadm --pool-pages-max DEFAULT:num_pages**
	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 are not available, Linux simply uses traditional sized pages.

Using huge pages for shared memory

To use huge pages for shared memory (journal buffers, replication journal pool and global buffers):

- Permit GT.M processes to use huge pages for shared memory segments (where available, FIS recommends option 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 the following command, which requires root privilges:

     ```
     echo gid >/proc/sys/vm/hugetlb_shm_group
     ```

- Set the environment variable HUGETLB_SHM for each process to "yes".

Using huge pages for GT.M process private memory

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 fully qualified path to libhugetlbfs.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).

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 libhugetlbfs (https://github.com/libhugetlbfs/libhugetlbfs) explains how and when to set LD_PRELOAD.

> **Note**
>
> - 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.

Configuring the Restriction facility

Post installation, a system administrator can optionally add a restrict.txt file in $gtm_dist to restrict the use of certain GT.M facilities to a group-name. The owner and group for $gtm_dist/restrict.txt can be different from those used to install GT.M. The file may contain zero or more of the following case-insensitive lines in any order:

```
APD_ENABLE:[comma-separated-list-of-options]:{path-to-sock-file|host:port}[:tls-id]
BREAK[:<group-name>]
CENABLE[:<group-name>]
DIRECT_MODE[:<group-name>]
DSE[:<group-name>]
HALT[:<group-name>]
LIBRARY:[<group-name>]
LKE:[<group-name>]
LKECLEAR:[<group-name>]
LOGDENIALS:[<group-name>]
PIPE_OPEN[:<group-name>]
TRIGGER_MOD[:<group-name>
ZBREAK[:<group-name>]
ZCMDLINE[:<group-name>]
ZEDIT[:<group-name>]
ZHALT[:<group-nam>]
ZSYSTEM[:<group-name>]
```

If the file $gtm_dist/restrict.txt does not exist, GT.M does not restrict any facilities.

Any non-empty lines that do not match the above format cause processes with read-only permission to behave as if they could not read the file, and GT.M enforces all restrictions.

Restrictions apply as follows:

GT.M facility	Behavior
APD_ENABLE	GT.M supports the ability to log actions initiated from a principal device including MUMPS commands typed interactively, or piped in by a script or redirect, from the principal device ($PRINCIPAL) and / or any

GT.M facility	Behavior
	information entered in response to a READ from $PRINCIPAL. An action initiated from $PRINCIPAL executes as usual when Audit Principal Device is disabled, which it is by default. However, when Audit Principal Device is enabled, GT.M attempts to send the action out for logging before acting on it. Additionally, the $ZAUDIT Intrinsic Special Variable (ISV) provides a Boolean value that indicates whether Audit Principal Device is enabled. Please see the Audit Principal Device section below for details.
BREAK	GT.M ignores any BREAK command
CENABLE	the process acts like $gtm_nocenable is TRUE and ignores any CENABLE deviceparameter
DIRECT_MODE	**mumps -direct** terminates immediately with a RESTRICTEDOP error
DSE	terminates immediately with a RESTRICTEDOP error
HALT	any HALT produces a RESTRICTEDOP error
LIBRARY	any attempt to load an external library produces a RESTRICTEDOP error Library restrictions apply to third party libraries loaded by GT.M directly and those loaded at the behest of GT.M applications. This restriction allows administrators to control which libraries GT.M is able to load at run-time. To allow library loading, place symbolic links to the desired libraries in $gtm_dist/plugin.
LKE	any invocation of the $gtm_dist/lke utility produces a RESTRICTEDOP error
LKECLEAR	any LKE CLEAR produces a RESTRICTEDOP error
LOGDENIALS	restrict logging when GT.M denies access to a process to the named groups GT.M supports logging a number of errors related to permissions and access using the syslog() facility. The GT.M restriction LOGDENIALS provides a facility for disabling this loggging on a Unix group basis. If this mechanism is not used the logging takes place for all GT.M processes. If the restriction is used logging takes place for specified groups only.
PIPE_OPEN	any OPEN of a PIPE device produces a RESTRICTEDOP error
TRIGGER_MOD	any $ZTRIGGER() or MUPIP TRIGGER that attempts a change or delete produces a RESTRICTEDOP error; in addition, while executing code within a trigger, ZBREAK results in a RESTRICTEDOP error, and both ZBREAK and ZSTEP actions are ignored
ZBREAK	any ZBREAK or ZSTEP produces a RESTRICTEDOP error
ZCMDLINE	GT.M returns an empty string for all references to $ZCMDLINE
ZEDIT	any ZEDIT produces a RESTRICTEDOP error
ZHALT	any ZHALT produces a RESTRICTEDOP error
ZSYSTEM	any ZSYSTEM produces a RESTRICTEDOP error

If the file exists, a process:

- that has write authorization to restrict.txt has no restrictions

- that has no read access to restrict.txt is restricted from all facilities for which GT.M supports a restriction (currently the above list)

- that has read-only access to restrict.txt is restricted from any listed facility unless it is a member of the group specified in the optional group-id following the facility name

Note that restricting $ZCMDLINE prevents things like: **mumps -run %XCMD 'for read x xecute x'** which can act as substitutes for Direct Mode.

In order to limit pathological looping from restricted HALT or ZHALT, if A GT.M process issues a second occurrence of the restricted command within half a second, the process terminates after sending a fatal error to both the principal device and the syslog, and also producing a GTM_FATAL* context file, but no core file. With these restrictions in place, a process should terminate with, for example: ZGOTO 0. Note that, with or without a restriction, executing these commands as part triggered logic on a replicating instance may cause the Update Server to terminate and thereby stop replication.

ZSYSTEM and PIPE OPEN command restriction facility

The GT.M restriction mechanism recognizes the following lines:

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

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

An example restrict file for this:

```
cat $gtm_dist/restrict.txt
ZSYSTEM_FILTER:^filterzsy
PIPE_FILTER:^filterzsy
```

The actual filter routine:

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

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

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

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

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

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

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

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

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

After the filter completes, GT.M restores the above to their values at the invocation of the filter.

Audit Logging facility

To enable the audit logging facility, add a line in the following format to $gtm_dist/restrict.txt:

```
A{PD|M}_ENABLE:[comma-separated-list-of-options]:{path-to-sock-file|host:port}[:tls-id]
```

APD_ENABLE enables the logging of all code entered from Direct Mode and optionally any input entered on the principal device ($PRINCIPAL). AM_ENABLE enables the logging of all MUPIP commands entered from the shell. With AM_ENABLE, MUPIP prompt is not allowed and produces the RESTRICTEDOP error.

- The optional "comma-separated-list-of-options" can consist of zero or more of these options:

  - TLS - Enables TLS connectivity between GT.M and the logger; this option requires the host information (e.g. IP/port or hostname/port)

  - RD - Enables logging of all responses READ from $PRINCIPAL in addition to that entered at the Direct Mode prompt. This option is more comprehensive and captures input that might be XECUTEd, but depending on your application architecture may significantly increase the amount of logged information. The RD option only applies when specifying APD_ENABLE.

- The "path-to-sock-file" is the absolute path of the UNIX domain socket file for connecting to the logger.

- The "host" is the hostname or numeric IPv4/IPv6 address of the logger; numeric IP addresses must be enclosed in square brackets (i.e. '[' and ']').

- The "port" is the port number the logger listens on.

- You can specify the same or different {path-to-sock-file|host:port} for any audit logging facility.

- The optional "tls-id" is the label of the section within the GT.M configuration file that contains TLS options and/or certificates for GT.M to use; APD ignores any "tls-id" if the "TLS" option is not specified.

If parsing any "A*_ENABLE" line in restriction file or initializing logger information fails, GT.M enforces all restrictions (default restriction file behavior).

Examples:

```
APD_ENABLE::/path/to/sock/file/audit.sock
```

Adding this line to the restriction file enables APD. GT.M connects with the logger via UNIX domain socket using the domain socket file "/path/to/sock/file/audit.sock" and sends all Direct Mode activity from $PRINCIPAL to logger.

```
AM_ENABLE::/path/to/sock/file/audit.sock
```

Adding this line to the restriction file enables the logging of all MUPIP commands from the shell. GT.M connects with the logger via UNIX domain socket using the domain socket file "/path/to/sock/file/audit.sock" and sends all MUPIP activity from the shell to logger. With AM_ENABLE, MUPIP prompt is not allowed and returns the RESTRICTEDOP error.

```
APD_ENABLE:RD:[123.456.789.100]:12345
```

Adding this line to the restriction file enables APD. GT.M connects with the logger (listening on port 12345 at the IPv4 address 1enable23.456.789.100) via TCP socket and sends all Direct Mode and READ activities from $PRINCIPAL to logger.

```
APD_ENABLE::loggerhost:56789
```

Adding this line to the restriction file enables APD. GT.M connects with the logger (listening on port 56789 at the hostname "loggerhost") using a TCP socket and sends all Direct Mode activities from $PRINCIPAL to logger.

```
APD_ENABLE:TLS,RD:[1234:5678:910a:bcde::f:]:12345:clicert
```

Adding this line to the restriction file enables APD. GT.M connects with the logger (listening on port 12345 at the IPv6 address 1234:5678:910a:bcde::f:) via TLS socket. GT.M configures its TLS options for APD based on the contents within the section of the configuration file labeled "clicert". GT.M sends all Direct Mode and READ activities from $PRINCIPAL to logger.

Logging

The "logger" is a separate server-like program responsible for receiving the to-be-logged information from GT.M and logging it. This separate program must be introduced by the user, either running in foreground or background, in order for logging to actually work. GT.M distributions include basic example logger programs.

The six fields in the message, separated by semicolons (';'), contain information on the to-be-logged activity. Each to-be-logged message sent to the logger from GT.M has the following format:

```
dist=<path>; src={0|1|2}; uid=<uid>; euid=<euid>; pid=<pid>;
▸ command=<text>
```

🔻

- The "dist" field, shows the path to location of the sender/user's $gtm_dist (GT.M executables).

- The "src" field shows zero (0) for input from unknown source, one (1) for Direct Mode input, two (2) for READ input from $PRINCIPAL, and three (3) for MUPIP commands.

- The next three fields ("uid", "euid", and "pid") show (respectively) decimal representations of the user ID, effective user ID, and process ID of the process that sent the message.

- The "command" field is the input provided on the GT.M side.

Examples:

```
dist=/path/to/gtm_dist; src=1; uid=112233445; euid=112233445; pid=987654; command=write "Hello world",!
dist=/usr/library/V123/dbg; src=2; uid=998877665; euid=998877665; pid=123456; command=set a=789
```

Click on 🔻 to download sample listener programs. You can also download *dm_audit_listener.zip* from http://tinco.pair.com/bhaskar/gtm/doc/books/ao/UNIX_manual/downloadables/dm_audit_listener.zip.

There are three different sample logger programs that can be used for logging. The difference amongst the three is the socket connection type used to communicate with GT.M. The dm_audit_unix_listener.c, dm_audit_tcp_listener.c, and dm_audit_tls_listener.c use the UNIX domain, TCP, and TLS socket type respectively to communicate with GT.M. Choose the logger program to use based on desired connection type.

Compiling Logger Program:

- The UNIX domain logger (dm_audit_unix_listener.c) can be compiled with:

```
gcc dm_audit_unix_listener.c -o dm_audit_unix
```

- The TCP logger (dm_audit_tcp_listener.c) can be compiled with:

```
gcc dm_audit_tcp_listener.c -o dm_audit_tcp
```

- To compile the TLS logger (dm_audit_tls_listener.c), you'll need to link the ssl (-lssl) and crypto (-lcrypto) libraries when compiling. Assuming you have OpenSSL development packege installed, the TLS logger can be compiled with:

```
gcc dm_audit_tls_listener.c -o dm_audit_tls -lssl -lcrypto
```

If you do not have the OpenSSL development package installed, then you will have to install it first.

Running Logger Program:

- The UNIX domain logger can be ran with:

```
./dm_audit_unix <logfile> <sockfile>
```

  - The "logfile" field is file path of output audit log.

  - The "sockfile" field is file path of unix domain socket file.

- The TCP logger can be ran with:

```
./dm_audit_tcp <logfile> <portno>
```

  - The "logfile" field is file path of output audit log.

  - The "portno" field is port number to listen on.

- The TLS logger can be ran with:

```
./dm_audit_tls <logfile> <portno> <certfile> <privkeyfile> <passphrase>
  [-clicert] [-cafile <CAfilepath>] [-capath <CApath>]
```

  - The "logfile" field is file path of output audit log.

  - The "portno" field is port number to listen on.

  - The "certfile" field is file path of TLS/SSL certificate to use.

  - The "privkeyfile" field is file path of private key to use.

  - The "passphrase" field is password or passphrase for certificate/key.

- The "clicert" option (optional), if present, then logger requests for client certificate when performing the TLS handshake with GT.M.

- The "CAfilepath" field (optional) is path to file of CA certificates in PEM format (specifies the locations for SSL context, at which CA certificates for verification purposes are located.)

- The "CApath" field (optional) is path to directory containing CA certificates in PEM format (specifies the locations for SSL context, at which CA certificates for verification purposes are located.)

Chapter 4. Global Directory Editor

Revision History		
Revision V7.0-001	24 November 2021	• In "Region Qualifiers" (page 71), add a table for equivalent settings for [NO]L[OCK_CRIT] in other utilities
Revision V7.0-000	12 February 2021	• In "Region Qualifiers" (page 71), correct default for LOCK_CRIT • In "Segment Qualifiers" (page 76), Update maximum value of Reserved Bytes
Revision V6.3-013	30 June 2020	• In "Global Directory Abbreviations" (page 54), correct typo: [HO] to [NO]
Revision V6.3-012	08 April 2020	• In "Examining the Default Global Directory" (page 52), add FBWR • In "SHow" (page 66), add FBWR • In "GDE Command Qualifier Summary" (page 86), add -FULLBLKWRT • In "Segment Qualifiers" (page 76), add -FULLBLKWRT qualifier.
Revision V6.3-011	20 December 2019	• In "Instance Qualifier" (page 83), add a link • In "Segment Qualifiers" (page 76), reflect higher extension_count
Revision V6.3-010	31 October 2019	• In "Logging" (page 42), Added additional documentation to the Direct Mode Auditing (or APD) feature. Specifically, more information on how to compile and run the logger programs is added. • In "GDE Command Summary" (page 84), add SHOW -COMMAND to table
Revision V6.3-007	04 February 2019	• In "Logging" (page 42), Downlodable dm_audit_listener.zip • In "Region Qualifiers" (page 71), make a correction • In "Segment Qualifiers" (page 76), correct max GLOBAL_BUFFER_COUNT; general fixes for the segment qualifiers summary table
Revision V6.3-006	26 October 2018	• In "Region Qualifiers" (page 71), add more information about epoch tapering and -NOSTATS. • In "Segment Qualifiers" (page 76), content improvements and corrections for ASYNCIO and DEFER_ALLOCATE.

Revision V6.3-005	03 July 2018	• In "Change" (page 62), correct the syntax of CHANGE -INSTANCE -FILE_NAME.
		• In "ZSYSTEM and PIPE OPEN command restriction facility" (page 40), add new section for the ZSYSTEM and PIPE OPEN command restriction facility
Revision V6.3-004	23 March 2018	• In "Region Qualifiers" (page 71), Add to stats qualifier description
Revision V6.3-003	12 December 2017	• In "Configuring the Restriction facility" (page 38), update TRIGGER_MOD; add HALT and ZHALT
		• In "Add" (page 59), fix section ordering
		• In "Change" (page 62), add CHANGE -INSTANCE -FILENAME
		• In "GDE Command Summary" (page 84), add entry for the CHANGE and SHOW -INSTANCE qualifier; properly order ALL and GBLNAME
		• In "Delete" (page 63), fix ordering of the format of the DELETE command
		• In "Rename" (page 65), fix the ordering of the format of the RENAME command
		• In "SHow" (page 66), add the -INSTANCE qualifier and improve the documentation of SHOW -COMMAND
		• In "Verify" (page 69), fix the ordering of the format of the VERIFY command
		• In "Instance Qualifier" (page 83), add new section
		• In "Segment Qualifiers" (page 76), correct the description of -extension_count and specify 262144 as the maximum LOCK_SPACE; remove redundant and misplaced Encryption text.
Revision V6.3-002	22 August 2017	• In "Configuring the Restriction facility" (page 38), add information about the Restriction facility
		• In "GDE Overview" (page 50), add statement on the return code
		• In "SHow" (page 66), clarify that SHOW -COMMAND does not display the current template settings.
		• In "Region Qualifiers" (page 71), change -JOURNAL qualifier example to match the new default value for BUFFER_SIZE.

Revision V6.3-001	20 March 2017	• In "Global Directory Abbreviations" (page 54), added AIO, AutoDB, Dall, Epoch Taper, LOCK Crit, and Stats abbreviations.
		• In "Examining the Default Global Directory" (page 52), added updates for V6.3-001.
		• In "SHow" (page 66), updated output for V6.3-001.
		• In "GDE Command Qualifier Summary" (page 86), updated the table for V6.3-001.
		• In "Region Qualifiers" (page 71), added a note about a condition when the number of attached processes exceed 32Ki on a QDBRUNDOWN enabled database; tweaked wording; alphabetize lists; added AUTODB, LOCK_CRIT, and STATS
		• In "Segment Qualifiers" (page 76), removed references to HP-UX and Solaris and various corrections and improvements; use delimiter before qualifiers; add -recover - forward for MM ; updated the summary table for V6.3-001 added ASYNCIO
Revision V6.2-002	17 June 2015	In "Segment Qualifiers" (page 76), added the [NO]EPOCHTAPER and -[NO]DEFER_ALLOCATE qualifiers.
Revision V6.1-000	01 August 2014	• In "Global Directory" (page 48), explained the concepts of mapping different subscripts of the same global to different regions.
		• In "Global Director Editor Commands" (page 58), added example of globals spanning regions, the description of the new -GBLNAME, -COLLATION, MUTEX_LOCK qualifiers.
Revision V6.0-003	27 January 2014	• In "Region Qualifiers" (page 71), corrected the syntax of [NO]STDNULLCOLL and -COLLATION_DEFAULT .
		• In "Segment Qualifiers" (page 76), added a paragraph about what happens when database file with automatic extension disabled (EXTENSION_COUNT=0) starts to get full.
Revision V6.0-001	27 February 2013	In "Segment Qualifiers" (page 76) 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 71), added the description of -[NO]INST[_FREEZE_ON_ERROR].

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 $ZBGLDIR at process startup from the environment variable gtmgbldir 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:
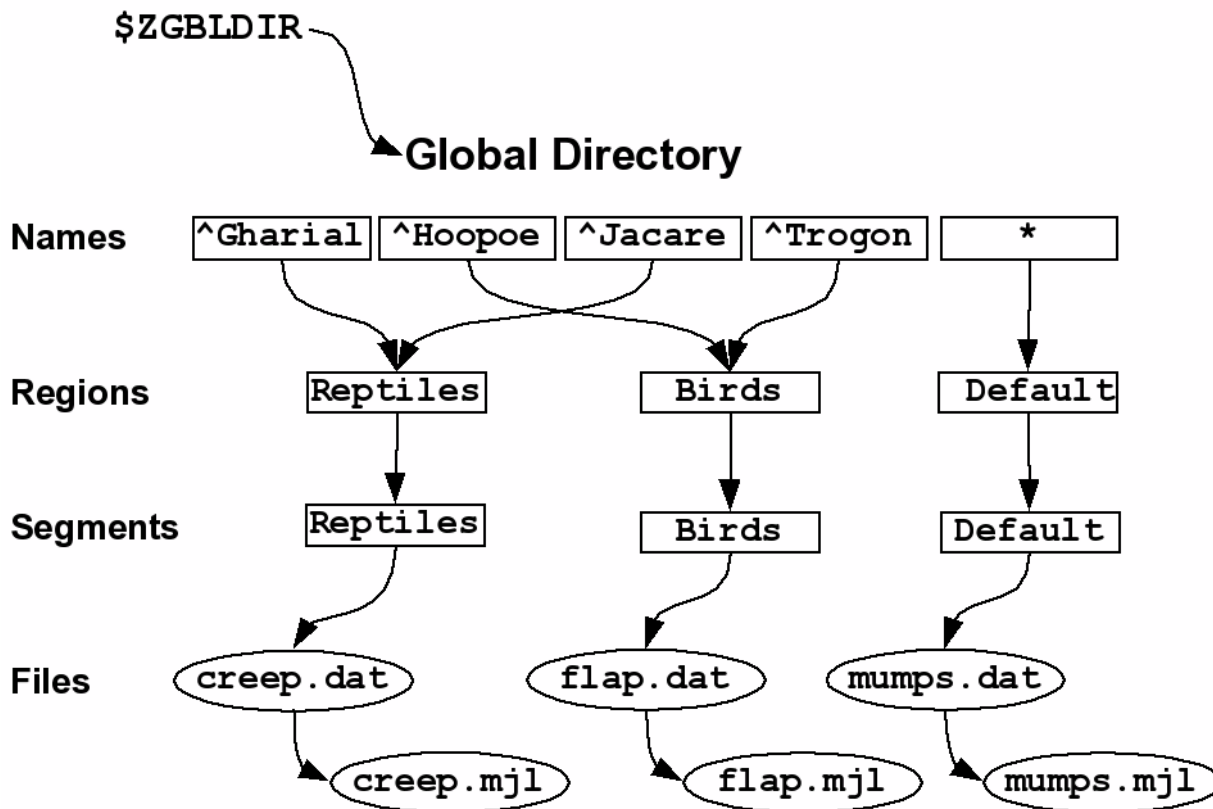
There are four M global variables--^Gharial, ^Hoopoe, ^Jacare, and ^Trogon. ^Gharial and ^Jacare map to region REPTILES that maps to database file creep.dat and ^Hoopoe and ^Trogon map to region BIRDS that maps to database file flap.dat. The default namespace * maps to a region called DEFAULT that maps 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	^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	

(right margin: Mapping, with downward arrow)

^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.

GDEOverview

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**. If you invoke it from the shell, GDE returns a status indicating success (0) or an issue (non-zero).

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 gtmgbldir
$ $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 ***
                                          Std     Inst
                       Def    Rec  Key Null   Null     Freeze Qdb  Epoch              LOCK
Region                 Coll   Size Size Subs   Coll Jnl on Err Rndwn Taper AutoDB Stats Crit
-------------------------------------------------------------------------------------------
<default>               0     256   64 NEVER   N    N   N    N     Y    N      Y    Sep
Segment      Active      Acc Typ Block    Alloc Exten   Options
-------------------------------------------------------------------------
<default>       *         BG  DYN  4096      100   100   GLOB =1024
                                                         LOCK =  40
                                                         RES  =   0
                                                         ENCR = OFF
                                                         MSLT =1024
                                                         DALL = YES
                                                         AIO  = OFF
                                                         FBWR =   0
<default>                 MM  DYN  4096      100   100   DEFER
                                                         LOCK =  40
                                                         MSLT =1024
                                                         DALL = YES
                                                         FBWR =   0
        *** NAMES ***
Global                      Region
-------------------------------------------------------------------------
*                           DEFAULT
```

```
                     *** REGIONS ***
                                                        Std      Inst
                     Dynamic                  Def   Rec  Key Null  Null     Freeze Qdb  Epoch                 LOCK
Region               Segment                  Coll  Size Size Subs Coll Jnl on Err Rndwn Taper AutoDB Stats Crit
---------------------------------------------------------------------------------------------------------------------
DEFAULT              DEFAULT                    0   256   64 NEVER  N   N   N      N     Y     N      Y    Sep
                     *** SEGMENTS ***
Segment              File (def ext: .dat)Acc Typ Block   Alloc Exten    Options
------------------------------------------------------------------------------------------
DEFAULT              mumps.dat       BG  DYN  4096    100  100     GLOB=1024
                                                                   LOCK=  40
                                                                   RES =   0
                                                                   ENCR= OFF
                                                                   MSLT=1024
                                                                   DALL= YES
                                                                   AIO = OFF
                                                                   FBWR=   0
                     *** MAP ***
  - - - - - - - - - - - Names - - - - - - - - - -
From                 Up to                       Region / Segment / File(def ext: .dat)
-------------------------------------------------------------------------------------------------
%                         ...                     REG = DEFAULT
                                                  SEG = DEFAULT
                                                  FILE = mumps.dat
LOCAL LOCKS                                       REG = DEFAULT
                                                  SEG = DEFAULT
                                                  FILE = mumps.dat
```

There are five primary sections in a Global Directory:

- TEMPLATES

- NAMES

- REGIONS

- SEGMENTS

- MAP

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
AIO                    -[NO]ASYNCIO
Alloc                  -ALLOCATION
AutoDB                 -[NO]AUTODB
AutoSwitch             -AUTOSWITCHLIMIT
Block                  -BLOCK_SIZE
Buff                   -BUFFER_SIZE
Dall                   -[NO]DEFER_ALLOCATE
Def Coll               -COLLATION_DEFAULT
Epoch Taper            -[NO]EPOCHTAPER
Exten                  -EXTENSION_COUNT
File                   -FILE_NAME
GLOB                   -GLOBAL_BUFFER_COUNT
Inst Freeze On Error   -[NO]INST_FREEZE_ON_ERROR
JNL                    -[NO]JOURNAL
Key Size               -KEY_SIZE
LOCK                   -LOCK_SPACE
LOCK Crit              -[NO]LOCK_CRIT
MSLT                   -MUTEX_SLOTS
Null Subs              -[NO]NULL_SUBSCRIPTS
Qdb Rndwn              -[NO]QDBRUNDOWN
Std Null Coll          -[NO]STDNULLCOLL
Rec Size               -RECORD_SIZE
RES                    -RESERVED_BYTES
Region                 -REGION
Stats                  -[NO[STATS
```

```
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 181).

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
```

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.

> **Note**
>
> Even when invoked from within GT.M, GDE always uses the gtmgbldir environment variable to identify its target

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 alphanumerics, 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 alphanumerics, 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.

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 MUPIP 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] -G[BLNAME] global-name [-GBLNAME-qualifier ...]
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
```

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
```

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
```

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 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. M 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. It may also alter the association of a global directory with a replication instance file.

The format of the CHANGE command is:

```
C[HANGE] -G[BLNAME] -C[OLLATION]=collation_number
C[HANGE] -I[NSTANCE] -F[ILE_NAME]={repl_inst_filename|""}
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...]
```

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

Once you exit GDE, mapping changes, including Instance mapping, 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] -G[BLNAME] global-name
D[ELETE] -N[AME] namespace
D[ELETE] -R[EGION] region-name
D[ELETE] -S[EGMENT] segment-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[XIT]
```

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] -G[BLNAME] old-global-name new-global-name
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
```

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

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

-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 must always appear after -COMMAND.

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.

SHOW -COMMAND displays the GDE commands for creating names, regions, and segments of the current global directory state in a target environment. However, it does not always include the same template settings (SHOW -TEMPLATE) of the current global directory. SHOW -COMMAND creates an appropriate set of templates that minimize other adjustments to recreate the current global directory. If the current GDE template settings (SHOW -TEMPLATE) are important for your application, you need set them again after applying the commands from GDE SHOW -COMMAND in the target environment.

> **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.

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

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

By default, SHOW displays -ALL.

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

-INSTANCE: Displays the current Instance Mapping, if any. This qualifier corresponds to the section of the SHOW report titled:

```
***INSTANCE***
```

-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***
```

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 ***
                                       Std   Inst
                         Def   Rec  Key Null  Null  Freeze Qdb  Epoch                 LOCK
Region                   Coll  Size Size Subs  Coll Jnl on Err Rndwn Taper AutoDB Stats Crit
--------------------------------------------------------------------------------------------
<default>                  0   256   64 NEVER   N   N   N    N     Y     N      Y    Sep
Segment       Active      Acc Typ Block   Alloc Exten  Options
--------------------------------------------------------------------------
<default>        *        BG  DYN  4096    100   100   GLOB =1024
                                                       LOCK =  40
                                                       RES  =   0
                                                       ENCR = OFF
                                                       MSLT =1024
                                                       DALL = YES
                                                       AIO  = OFF
                                                       FBWR =   0
<default>                 MM  DYN  4096    100   100   DEFER
                                                       LOCK =  40
                                                       MSLT =1024
                                                       DALL = YES
                                                       FBWR =   0
```

This displays only the TEMPLATES section of the Global Directory.

```
GDE>SHOW -command
TEMPLATE -REGION -NOAUTODB
TEMPLATE -REGION -COLLATION_DEFAULT=0
TEMPLATE -REGION -EPOCHTAPER
TEMPLATE -REGION -NOINST_FREEZE_ON_ERROR
TEMPLATE -REGION -JOURNAL=(ALLOCATION=2048,AUTOSWITCHLIMIT=8386560,BEFORE_IMAGE,BUFFER_SIZE=2312,EXTENSION=2048)
TEMPLATE -REGION -KEY_SIZE=64
TEMPLATE -REGION -NOLOCK_CRIT
TEMPLATE -REGION -NULL_SUBSCRIPTS=NEVER
TEMPLATE -REGION -NOQDBRUNDOWN
TEMPLATE -REGION -RECORD_SIZE=256
TEMPLATE -REGION -STATS
TEMPLATE -REGION -NOSTDNULLCOLL
!
TEMPLATE -REGION -NOJOURNAL
!
TEMPLATE -SEGMENT -ACCESS_METHOD=BG
TEMPLATE -SEGMENT -ALLOCATION=100
TEMPLATE -SEGMENT -NOASYNCIO
TEMPLATE -SEGMENT -BLOCK_SIZE=4096
TEMPLATE -SEGMENT -DEFER_ALLOCATE
TEMPLATE -SEGMENT -NOENCRYPTION_FLAG
TEMPLATE -SEGMENT -EXTENSION_COUNT=100
TEMPLATE -SEGMENT -FULLBLKWRT=0
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 -NOASYNCIO
TEMPLATE -SEGMENT -BLOCK_SIZE=4096
TEMPLATE -SEGMENT -DEFER
TEMPLATE -SEGMENT -DEFER_ALLOCATE
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] -A[LL]
V[ERIFY] -G[BLNAME]
V[ERIFY] -M[AP]
V[ERIFY] -N[AME] [namespace]
V[ERIFY] -R[EGION] [region-name]
V[ERIFY] -S[EGMENT] [segment-name]
V[ERIFY] -T[EMPLATE]
```

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

-ALL   Checks all map and template data.

-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.

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**
-R[EGION]=region-name (characters)	(none)	1A	16A/N

Region Qualifiers

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

-[NO]AU[TODB]

Specifies whether GT.M should implicitly create a database file for the region if none exists when a process attempts to access it. Because it carries lower operational risk and provides better operational control, the common practice is to create database files with MUPIP CREATE. However, AUTODB may simplify operations when you have scratch or temporary databases which are best deleted and recreated as a part of standard operation procedures.

The default is NOAUTODB.

-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.

-[NO]EPOCHTAPER

Tries to minimize epoch duration by reducing the number of buffers to flush by GT.M and the file system (via an fsync()) as the epoch (time-based or due to a journal file auto-switch) approaches. By default, EPOCHTAPER is enabled. Epoch tapering reduces the impact of I/O activity during an epoch event. Application that experience high load and/or need to reduce latency may benefit from epoch tapering.

-[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 254).

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 297).

-[NO]J[OURNAL][=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 181).

The journal-option-list includes:

- A[LLOCATION]=blocks

- AUTOSWITCHLIMIT=blocks

- [NO]BE[FORE_IMAGE]

- BU[FFER_SIZE]=pages

- E[XTENSION]=blocks

- F[ILE_NAME]=file-specification-name

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 181).

-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 MUPIP SET to explicitly turn it ON, and you must specify BEFORE/NOBEFORE at that time.

Example:

```
CHANGE -REGION DEFAULT -JOURNAL=(ALLOCATION=2048,AUTOSWITCHLIMIT=8386560,BEFORE_IMAGE,BUFFER_SIZE=2312,EXTENSION=2048)
```

For information on all Journal options and their allowable minimum and maximum values, see "SET -JOURNAL Options " (page 197) in the "GT.M Journaling" chapter.

Summary

-K[EY_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

-[NO]L[OCK_CRIT]

GT.M uses critical sections for LOCK management and various database activities. LOCK_CRIT in GDE specifies that GT.M should use separate critical sections for LOCK management and database activity. NOLOCK_CRIT specifies the use of the database critical section for both LOCK and database activity.

Because, in the current implementation, FIS has not identified any reason to share resource management between LOCKs and database actions, we have no recommendations other than to choose what seems to work better for your application.

By default, GDE uses LOCK_CRIT - separate resource management for LOCKs and database actions.

The -[NO]L[OCK_CRIT] setting appears with a different name in other utilities and report output. The following table shows the equivalent setting of the [NO]L[OCK_CRIT] GDE qualifier in other utilities:

Utility/Report	LOCK and DB share the SAME critical section	LOCK and DB have SEPARATE critical section (default)
GDE Region Qualifier	**-NOLOCK_CRIT**	**-LOCK_CRIT**
DSE DUMP - FILEHEADER	**LOCK shares DB critical section : TRUE**	**LOCK shares DB critical section : FALSE**
MUPIP SET	**-LCK_SHARES_DB_CRIT**	**-NOLCK_SHARES_DB_CRIT**
MUPIP DUMPFHEAD	**("sgmnt_data.lock_crit_with_db")=1**	**("sgmnt_data.lock_crit_with_db")=0**

-[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]Q[DBRUNDOWN]

Shortens normal process shutdown when a large number of processes accessing a database file need to shutdown almost simultaneously, for example, in benchmarking scenarios or emergencies.

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 that the last process to exit might leave the database shared memory and IPC resources in need of cleanup. Except after the number of concurrent processes exceeds 32Ki, QDBRUNDOWN minimizes the prossibility of abandoned resources, but it cannot eliminate it. When using QDBRUNDOWN, use an explicit MUPIP RUNDOWN of the database file after the last process exits, to ensure the cleanup of database shared memory and IPC resources; not doing so risk database damage.

When a database has QDBRUNDOWN enabled, if the number of attached processes ever exceeds 32Ki, GT.M stops tracking the number of attached processes, which means that it cannot recognize when the number reaches zero (0) and the shared resources can be released. The process that detects this event issues a NOMORESEMCNT in the system log. This means an orderly, safe shutdown requires a MUPIP JOURNAL -ROLLBACK -BACKWARD for replicated databases, a MUPIP JOURNAL -RECOVER -BACKWARD for unreplicated journaled databases and a MUPIP RUNDOWN for journal-free databases.

-R[ECORD_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][STA[TS]

Specifies whether GT.M should permit processes to share their database access statistics for other processes to monitor. When on, this characteristic causes GT.M to create a small MM database for the associated region to hold the shared statistics. There may be operational or security reasons to prohibit sharing of statistics. For example, GT.M does not share statistics on database files that exist solely to support GT.M features.

Note that a process disables itself from maintaining the shared statistics when it fails to open a statsDB. It does not, however, disable subsequently starting processes from maintaining the shared statistics.

By default, GDE uses STATS.

For more information, refer to VIEW "[NO]STATSHARE" and ^%YGBLSTAT in GT.M Programmer's Guide and gtm_statshare and gtm_statsdir in "Environment Variables" (page 19).

-[NO]STD[NULLCOLL]

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.

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
-[NO]AU[TODB]	Disabled	-	-
-C[OLLATION_DEFAULT]=number (integer)	0	0	255
-D[YNAMIC_SEGMENT] =segment-name (char)	-	1	31
-[NO]EPOCHTAPER	ENABLED	-	-
-[NO]INST[_FREEZE_ON_ERROR]	DISABLED	-	-
-[NO]J[OURNAL] [=journal-option-list]	-NOJ	-	-

GDE REGION Qualifiers			
QUALIFIER	DEFAULT	MINIMUM	MAXIMUM
-K[EY_SIZE]=size in bytes (integer)	64	3	1,019
-[NO]L[OCK_CRIT]	Disabled (not shared)	-	-
-N[ULL_SUBSCRIPTS]=[ALWAYS|NEVER|EXISTING]	NEVER	-	-
-[NO]Q[DBRUNDOWN]	Disabled	-	-
-R[ECORD_SIZE]=size in bytes (integer)	256	0	1,048,576 (1 MiB)
-[NO]STA[TS]	ENABLED	-	-
-[NO]STD[NULLCOLL]	No	-	-

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 181).

  - 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 88).

  - 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 supports MUPIP JOURNAL -RECOVER -FORWARD and MUPIP JOURNAL -ROLLBACK -FORWARD. With MM, MUPIP JOURNAL -RECOVER -BACKWARD only generates lost and broken transaction files but cannot recover the database.

  - Depending on your file system, MM may be an option when you need performance advantage in situations where the above restrictions are acceptable.

- 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.

- 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.

--[NO]AS[YNCIO]

Determines whether an access method BG database file uses asynchronous I/O rather than using synchronous I/O through the file system cache.

With ASYNCIO, GT.M assumes responsibility for writing database updates directly to secondary storage, essentially bypassing the file system and its cache. This can yield improved behavior if the file system has trouble handling GT.M database I/O, particularly file synchronization (fsync). ASYNCIO eliminates some memory activities and may improve performance in some configurations.

Some notes and observations:

- As asynchronous IO dispenses with the UNIX file buffer cache, GT.M global buffers are the sole caching mechanism. To make asynchronous IO perform well, you will likely need to increase the number of global buffers considerably. Assign adequate database global buffers to compensate for async I/O bypassing the file system cache. With GT.M's limit of 2GiB per shared memory segment, a database segment with 4KiB blocks has a limit of almost two million global buffers.

- A large number of global buffers potentially implies a large number of dirty global buffers to be flushed at an epoch. You should investigate the impact on application response time of GT.M epoch tapering vs. turning off epoch tapering and using a separate stand-alone process that executes a line of code such as: **for set x="" for set x=$view("gvnext",x) quit:""=x view "dbflush":x,"dbsync":x,"epoch":x hang** *n* where *n* is a number that causes each region to be flushed at an appropriate interval. If you choose this option, remember to turn off epoch tapering, and to set the epoch interval in the file header to be large enough to prevent application processes from performing epochs, and consider scripted timely switching of journal files by other than application processes (switching journal files involves an epoch).

- On AIX, consider mounting file systems with the CIO mount option. The CIO mount option drops support for the file buffer cache (unused by asynchronous IO), and also eliminates a lock that is a potential bottleneck to GT.M performance on the AIX jfs2 filesystem.

- If a process encounters a situation where it needs to perform an asynchronous write, but has no available slots with which to manage an additional one, it either falls back to synchronous writing if the write is blocking other actions, and otherwise defers the write until a slot becomes available as other writes complete. Linux allocates the structures on a system-wide basis with the setting of /proc/sys/fs/aio-max-nr. FIS recommends setting /proc/sys/fs/aio-max-nr to 1048576.

- For Linux x86_64, set the environment variable gtm_aio_nr_events: the gtm_aio_nr_events environment variable controls the number of structures a process has per global directory to manage asynchronous writes, and therefore determines the number of concurrent writes a process can manage across all regions within a global directory. If not specified, the value controlled by gtm_aio_nr_events defaults to 128. If a process encounters a situation where it needs to perform an asynchronous write, but has no available slots with which to manage an additional one, it either falls back to synchronous writing if the write is blocking other actions, and otherwise defers the write until a slot becomes available as other writes complete. The default value for gtm_aio_nr_events (that is, 128) should be sufficient for most applications. Change the value for the gtm_aio_nr_events environment variable based on benchmarking.

- Monitor the number of database writes errors for each global directory with set x="" for set x=$view("gvnext",x) quit:""=x $$^%PEEKBYNAME("sgmnt_data.wcs_wterror_invoked_cntr",x). If there are database write errors, your application may benefit from altering the number of gtm_aio_nr_events.

- While database write errors may indicate a problem with database writes if your storage system is starting to degrade, in a well-functioning environment, they indicate that a write attempt was unable to start an asynchronous write because it was unable to obtain a free resource of the type associated with /proc/sys/fs/aio-max-nr and gtm_aio_nr_events. In such a case, GT.M either defers the write in hopes that a resource will come available for a future attempt, or, if the write is blocking the application, GT.M performs a synchronous direct I/O. A synchronous direct I/O tends to lengthen response times and reduce throughput, so if you see non-trivial counts of such errors, you should revisit your settings for the resource.

- Limited experience with solid-state storage (SSDs) on Linux in the GT.M development environment suggests a considerable difference in asynchronous IO performance on the same underlying hardware, with f2fs performing better than xfs, which in turn performed better than ext4.

While there is reason to hope that ASYNCIO can provide better and more uniform performance, to this point we have limited information on performance comparisons, so FIS recommends well thought out benchmarking of your application in a suitable test environment. Please consider the above observations in this light.

By default GDE uses NOASYNCIO. On segments with an access method of MM, GT.M ignores this setting.

-BL[OCK_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 up 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]DEFER_ALLOCATE

With -DEFER_ALLOCATE, GT.M instructs the file system to create the database file as a sparse file. Before using -DEFER_ALLOCATE, ensure that your underlying file system supports sparse files. By default UNIX file systems, and GT.M, use sparse (or lazy) allocation, which defers actual allocation until blocks are first written.

- Utilities such as du report typically show lower disk space usage for a database file with -DEFER_ALLOCATE because GT.M instructs the file system to defer disk space allocation to the time when there is an actual need. With -NODEFER_ALLOCATE, such utilities report higher disk space usage count as GT.M instructs the file system to preallocate disk space without waiting for a need to arise.

- -DEFER_ALLOCATE makes database file extensions lighter weight. However, disk activity may tend towards causing fragmentation.

- To switch an existing database file so it immediately preallocates all blocks, first use MUPIP SET -NODEFER_ALLOCATE to set the switch in the database file header, followed by **MUPIP EXTEND -BLOCKS=n**, where **n >= 0**. Failures to preallocate space produce a PREALLOCATEFAIL error.

- The default is DEFER_ALLOCATE.

-[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 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 1,048,575 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.

-FU[LLBLKWRT]={0|1|2}

Determines whether GT.M writes only valid database blocks contents, or a full block including meaningless trailing content. Full block writes are more efficient with some secondary storage because they avoid read-before-write.

- When -FULLBLKWRT=2, a process writes all newly allocated database blocks in their entirety regardless of their actual valid contents. This relieves some file systems from tracking as much unallocated space and thus reduces file system metadata maintenance.

- When -FULLBLKWRT=1, a process writes entire file system blocks in their entirety regardless of their actual valid contents, on some file systems, this avoids reading in advance of most writes and thus reduces file system load and increases response time.

- When -FULLBLKWRT=0 (the default), a process writes only valid data. FIS advises using the same value of -FULLBLKWRT for all processes.

> ### Note
>
> When the file system block size and the database block size are the same there is no difference between the settings of 1 and 2.

-G[LOBAL_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 32 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 2,097,151 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 globals buffers that are "paged out."
>
> 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.

-L[OCK_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.

The maximum LOCK_SPACE is 262,144 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.

-R[ESERVED_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 = Block Size - Key Size - 40(block/record header overheads)

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	-	-
-AL[LOCATION]=size (blocks)	100	10	1,040,187,392(992Mi)
-[NO]AS[YNCIO]	FALSE	-	-
-BL[OCK_SIZE]=size (bytes)	1,024	512	65,024
-[NO]DEFER_[ALLOCATE]	TRUE	-	-
-[NO]E[NCRYPTION]	FALSE	-	-
-EX[TENSION_COUNT]=size (blocks)	100	0	1,048,575
-F[ILE_NAME]=file-name (chars)	mumps.dat	-	255
-FU[LLBLKWRT]={0|1|2}	0	0	2
-G[LOBAL_BUFFER_COUNT]=size (blocks)	1024	64	2,097,151*
-L[OCK_SPACE]=size (pages)	40	10	262,144
-M[UTEX_SLOTS]=integer	1,024	64	32,768
* May vary by platform			

GDE SEGMENT Qualifiers			
QUALIFIER	DEFAULT	MIN	MAX
-R[ESERVED_BYTES]=size (bytes)	0	0	Block Size - Key Size - 32
* **May vary by platform**			

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

Instance Qualifier

The following -INSTANCE qualifier is used with the CHANGE command.

-F[ILE_NAME=[repl_inst_filename|""]

- -FILE_NAME=repl_inst_filename maps a replication instance file with the global directory. -FILE_NAME="" removes the mapping of a global directory with a replication instance file.

- When a global directory is use, the mapping set with CHANGE -INSTANCE FILE_NAME=repl_inst_filename overrides any setting of the gtm_repl_instance environment variable. However, other utilities (MUPIP, LKE, and DSE) use the setting of the gtm_repl_instance environment variable.

For more information, refer to

GDE CommandSummary

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]	-G[BLNAME]	global-name -C[OLLATION]=collation
-	-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...]
C[HANGE]	-G[BLNAME]	global-name -C[OLLATION]=collation
-	-I[NSTANCE]	replication-instance -F[ILE_NAME]=replication_instance_file
-	-N[AME]	namespace -R[EGION]=new-region
-	-R[EGION]	region-name [-REGION-qualifier...]
-	-S[EGMENT]	segment-name [-SEGMENT-qualifier]
D[ELETE]	-G[BLNAME]	global-name -C[OLLATION]=collation
-	-N[AME]	namespace
-	-R[EGION]	region-name
-	-S[EGMENT]	segment-name
E[XIT]	N/A	N/A
HE[LP]	N/A	Keyword
* -ALL is the default for the SHOW and VERIFY commands.		

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

GDE Command Summary		
Command	**Specified Object Type**	**Required Object Name/[Optional] Qualifier**
-	-M[AP]	N/A
-	-T[EMPLATE]	N/A
* -ALL is the default for the SHOW and VERIFY commands.		

GDE CommandQualifier 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
-[NO]AS[YNCIO]	FALSE	-	-	-	-	X
-[NO]AU[TODB]	FALSE	-	-	-	X	-
-BL[OCK_SIZE]=size(bytes)	1024	512	65024	-	-	X
-C[OLLATION_DEFAULT]=id-number(integer)	0	0	255	-	X	-
-[NO]DEFER_ALLOCATE	TRUE	-	-	-	-	X
-D[YNAMIC_SEGMENT]=segment-name(chars)	*	1A	16A/N	-	X	-
-[NO]ENCRYPTION	FALSE	-	-	-	-	X
-[NO]EPOCHTAPER	TRUE	-	-	-	-	X
-EX[TENSION_COUNT]=size(blks)	100	0	65535	-	-	X
-F[ILE_NAME]=file-name (chars)	**	1A	255A/N	-	-	X
-FU[LLBLKWRT]={0|1|2}	0	0	2	-	-	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	-
* DEFAULT is the default region- and segment-name						
** MUMPS is the default file-name						
*** May vary by platform						
**** -NONULL_SUBSCRIPTS						

GDE Command Qualifiers						
QUALIFIER	DEF	MIN	MAX	NAM	REG	SEG
-[NO]L[OCK_CRIT]	FALSE	-	-	-	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[OURNAL]=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

* DEFAULT is the default region- and segment-name

** MUMPS is the default file-name

*** May vary by platform

**** -NONULL_SUBSCRIPTS

Chapter 5. General Database Management

Revision History		
Revision V7.0-004	20 September 2022	• In "-RETry " (page 103), specify that -RETRY=count is the number of retry attempts. • In "-Region" (page 106), autodb • In "-V6" (page 106), MUPIP CREATE -V6 and the env var • In "CREATE" (page 106), add MUPIP CREATE -V6; Note about AutoDBs • In "-Truncate" (page 144), Update TRUNCATE notes for standalone
Revision V7.0-003	24 June 2022	• In "-Region" (page 127), specify that INTEG processes regions based on the order specified in the region-list • In "-REGION=region" (page 108), specify that DUMPFHEAD selects regions based on the order specified in the region-list.
Revision V7.0-002	29 March 2022	• In "BACKUP" (page 97), correct the syntax of -NOONLINE; specified that MUPIP BACKUP selects regions based on the user-specified order • In "-PROblksplit" (page 157), add new section
Revision V7.0-001	24 November 2021	• In "-Record" (page 102), added a note of the added timestamping functionality of -RECORD. • In "-RETry " (page 103), add new section • In "-Lck_shares_db_crit" (page 156), add reference to the -[NO]L[OCK_CRIT] section. • In "BACKUP" (page 97), add -RETRY and -SHOWPROGRESS • In "-FLUSH" (page 108), add clarifications • In "SET" (page 150), added -STATSD[B_ALLOCATION] qualifier; corrected ordering of LCK_SHARES_DB_CRIT • In "MUPIP Command Summary" (page 170), The -STATSD[B_ALLOCATION] qualifier expects to get an integer. • In "-StatsDB_allocation " (page 159), change the minimum statsdb_allocation value to 128.

Revision V7.0-000	12 February 2021	• In "MUPIP Command Summary" (page 170), removed -DBG as it is for internal FIS use. • In "-REServed_bytes" (page 158), Update maximum Reserved Bytes
Revision V6.3-014	06 October 2020	• In "FTOK " (page 120), add new qualifiers and functionality. • In "SEMAPHORE" (page 150), add SEMAPHORE command
Revision V6.3-013	30 June 2020	• In "BACKUP" (page 97), Indicate that "-B" is not sufficient to indicate a "-BYTESTREAM" backup. It must be at least "-BY". • In "-[NO]A[UTORELEASE" (page 117), specify that GT.M does not honor MUPIP SET commands even when -AUTORELEASE is set. • In "Operations - Standalone and Concurrent Access" (page 96), remove a redundant and add a reference to the MUPIP Command Summary table. • In "SET" (page 150), Remove a bullet point about standalone access. • In "MUPIP Command Summary" (page 170), add a new Standalone Access column; correct note about EXTRACT; change -B[YTESTREAM] to -BY[TESTREAM]; change STD[NULLSUBS] to STD[NULLCOLL]; remove SPIN[_SLEEP_LIMIT]
Revision V6.3-012	08 April 2020	• In "RESTORE" (page 147), added an example and made content improvements. • In "SET" (page 150), add -FULLBLKWRT • In "SIZE" (page 161), correct and expand the subscript documentation. • In "MUPIP Command Summary" (page 170), add -FULLBLKWRT • In "GDE Command Qualifier Summary" (page 86), add -FULLBLKWRT
Revision V6.3-011	20 December 2019	• In "SIZE" (page 161), enhance the 2 sigma explanation
Revision V6.3-010	31 October 2019	• In "REORG " (page 136), add -DOWNGRADE and -UPGRADE. • In "SIZE" (page 161), mupip size -subscript changes • In "MUPIP Command Summary" (page 170), add qualifiers for REORG; add -CORRUPTDB

Revision V6.3-009	27 June 2019	• In "REORG " (page 136), Add -keep, -nocoalesce, -nosplit, -noswap
		• In "MUPIP Command Summary" (page 170), add entries for -GVPATFILE and -STOPRECEIVERFILTER.
		• In "-Read_only" (page 158), add a note on using -NOREAD_ONLY.
Revision V6.3-008	24 April 2019	• In "SET" (page 150), add information about the -[NO]ENCYRPTABLE and -ENCRYPTIONCOMPLETE qualifiers
Revision V6.3-007	04 February 2019	• In "BACKUP" (page 97), Add caution regarding NOPREVLINK option of -NEWJNLFILE qualifier
		• In "SET" (page 150), add -trigger_flush and writes_per_flush
		• In "MUPIP Command Summary" (page 170), add [NO] for LOSTTRANS and BROKENTRANS qualifiers
		• In "-Encrypt" (page 138), add a note about the FIS recommendation for rotating the encryption key of your database.
		• In "-Global_buffers" (page 154), The max global buffers is 2,097,151
Revision V6.3-006	26 October 2018	• In "-NETtimeout" (page 101), add RESTORE as a compatible MUPIP command.
		• In "-FOrmat" (page 112), improve the description of the GO format.
		• In "FREEZE " (page 115), tweaks for clarity
		• In "-[NO]A[UTORELEASE" (page 117), fix typo
		• In "-ONLine" (page 118), correct typos
		• In "RESTORE" (page 147), change file-list to bytestrm-bkup-list, add -NETTIMEOUT syntax and reference, and improve the description.
		• In "SET" (page 150), remove -REPLICATION from the destination list,-STANDALONENOT and -PARTIAL_RECOV_BYPASS, and minor changes to the definition of MUPIP SET.
		• In "-ACcess_method " (page 152), fix a typo.
		• In "-DEFER_Allocate " (page 152), improve the description of DEFER_ALLOCATE.
		• In "-Epochtaper" (page 153), add more information about epoch tapering.

		• In "-REPlication" (page 152), remove -FILE and -REGION from the list of incompatible qualifiers. • In "-STATs " (page 159), specify that a process disables itself from maintaining shared statistics when it fails to open a statsdb.
Revision V6.3-005	03 July 2018	• In "-Database" (page 101), remove reference to magnetic tape • In "BACKUP" (page 97), change "breaking a disk mirror" to "disk mirroring". • In "EXTRACT" (page 111), remove reference to magnetic tape, add -NULL_IV, and corrections for -SELECT.
Revision V6.3-004	23 March 2018	• In "Examples for MUPIP REORG" (page 144), add a simple REORG -ENCRYPT example to encrypt an unencrypted database. • In "-Null_iv" (page 113), correct qualifier format and add missing information. • In "-Select" (page 114), change prefix to suffix; add clarity about using suffixes; remove the point about supporting parenthetical list such as (a,B,C). • In "CREATE" (page 106), Max # of blocks is 992Mi • In "MUPIP Command Summary" (page 170), remove the entry for -updhelper • In "-Encrypt" (page 138), made minor refinements and add the requirement of an an encryption setup for -ENCRYPTABLE. • In "-Read_only" (page 158), note about semaphores • In "-STATs " (page 159), Expand information on STATS • In "-STDnullcoll " (page 160), change STATs to STDnullcoll and remove the duplicate STDnullcoll section
Revision V6.3-003	12 December 2017	• In "-Hard_spin_count" (page 155), improve the description. • In "BACKUP" (page 97), change -COMPREHENSIVE to -DATABASE • In "FREEZE " (page 115), add information about DBFREEZEON and DBFREEZEOFF messages that MUPIP FREEZE sends to the system log.

		• In "Operations - Standalone and Concurrent Access" (page 96), remove redundant entry for MUPIP BACKUP
		• In "SET" (page 150), add entries for -Null_subscripts, -Stdnullcoll, and -Read_only qualifiers.
		• In "MUPIP Command Summary" (page 170), add a new GT.M Version column - maintainable since GT.M V5.4-002.
		• In "-Lock_space" (page 156), specify 262144 as the maximum LOCK_SPACE.
		• In "-Read_only" (page 158), add conflict handling explanation
		• In "-Spin_sleep_mask" (page 159), add new section.
Revision V6.3-002	22 August 2017	• In "SET" (page 150), correct abbreviations for -SLEEP_SPIN_COUNT and -SPIN_SLEEP_LIMIT.
Revision V6.3-001	20 March 2017	• In "-REPLInstance " (page 103), admonished that the source server needs to be started at least once before taking a backup of the replication instance file.
		• In "Examples for MUPIP INTEG" (page 129), added the explanation of % Used
		• In "Examples for MUPIP REORG" (page 144), added an example for MUPIP REORG.
		• In "-FOrmat" (page 112), added changes for -null_iv qualifier.
		• In "-Null_iv" (page 113), tweaked wording
		• In "-Stats" (page 127), added STATS qualifier
		• In "-FOrmat" (page 132), added enhancements related to MUPIP LOAD -FORMAT.
		• In "-Onerror" (page 134), corrected the default and qualifier item description
		• In "-Stdin " (page 134), added an Index entry
		• In "BACKUP" (page 97), removed d-DBG from MUPIP BACKUP qualifier list as it is for FIS internal use.
		• In "DOWNGRADE" (page 107), added information about the V63000A option and a note about performing MUPIP INTEG -ONLINE prior to downgrading a database file.

		• In "DUMPFHEAD" (page 108), added information about the MUPIP DUMPFHEAD command
		• In "FREEZE " (page 115), added information about the [NO]ONLINE and [NO]AUTORELEASE qualifiers.
		• In "Operations - Standalone and Concurrent Access" (page 96), added a note about MUUSERLBK and MUUSERECOV errors.
		• In "REORG " (page 136), -REGION= is a syntax error.
		• In "RUNDOWN" (page 148), added references to statistics database files
		• In "SET" (page 150), added the SPIN_SLEEP_LIMIT,SLEEP_SPIN_COUNT, ASYNCIO, LCK_SHARES_DB_CRIT, and STATS qualifiers.
		• In "SIZE" (page 161), added explanation for the 2 Sigma column of the report
		• In "TRIGGER" (page 164), added caution about relying on journaling of trigger - upgrade.
		• In "MUPIP Command Summary" (page 170), added FREEZE, DOWNGRADE, DUMPFHEAD commands and MUPIP INTEG and SET qualifiers for V6.3-001
		• In "GDE Command Qualifier Summary" (page 86), updated the table for V6.3-001.
		• In "-Encrypt" (page 138), added information about the -ENCRYPT qualifier; Added a note that reorg -encrypt does not support changing of encryption algorithms.
		• In "-DEFER_Allocate " (page 152), removed references to HP-UX and Solaris.
		• In "-Epochtaper" (page 153), replaced "QDBRUNDOWN" with "-EPOCHTAPER"
		• In "-Qdbrundown" (page 157), added a note about the condition when the number of attached processes exceeds 32Ki.
Revision V6.2-002	17 June 2015	• In "SET" (page 150), added the [NO]EPOCHTAPER and -[NO]DEFER_ALLOCATE qualifiers.
		• In "INTEG " (page 121), added a recommendation against running ONLINE INTEG as the same time as MUPIP REORG.
		• In "RUNDOWN" (page 148) and "FREEZE " (page 115), specified that a successful rundown turns off any MUPIP FREEZE.

Revision V6.2-001	27 February 2015	• In "LOAD" (page 131), added the new ONERROR qualifier and information about automatically detecting file format (BINARY/ZWR/GO). • In "EXTRACT" (page 111), added the REGION qualifier. • In "SIZE" (page 161) and "INTEG " (page 121) , added information about adjacency reporting. • In In "TRIGGER" (page 164), added information about return status, LGTRIG journal records, and specified that MUPIP TRIGGER -SELECT works even if a multi-line XECUTE string does not terminate with a newline character. • In "TRIGGER" (page 164), added trigger expression source line limit, the UPGRADE qualifier, and a note about how MUPIP TRIGGER operates within a TP transaction. • In "Introduction" (page 95), specified that GT.M processes performing read operations can access databases on a filesystem mounted read-only. • Added the new "RCTLDUMP" (page 135) command. • Added the new "-Relinkctl" (page 150) qualifier for MUPIP RUNDOWN.
Revision V6.1-000	01 August 2014	• In "Introduction" (page 95), added a note recommending against runing any GT.M component as root.
Revision V6.0-003	27 January 2014	• In "SET" (page 150), added the -Mutex_slots qualifier. • In "RUNDOWN" (page 148), added the -Override qualifier.
Revision V6.0-001/2	10 April 1013	In "BACKUP" (page 97), added notes for "Before starting a MUPIP BACKUP".
Revision V6.0-001/1	22 March 1013	Improved the formatting of all command syntaxes.
Revision V6.0-001	27 February 2013	• In "BACKUP" (page 97), added information about the BKUPRUNNING message and added information on the handling of KILLs in Progress and Abandoned Kills during a backup. • In "EXTRACT" (page 111), added the description of -STDOUT. • In MUPIP INTEG, specified the threshold limit of incorrectly marked busy errors in "-MAP" (page 125) and added information

		on clearing KILLs in Progress and Adandoned Kills.
Revision V6.0-000/1	21 November 2012	• In MUPIP SET, added the description of "-PArtial_recov_bypass" (page 157). • Updated "TRIGGER" (page 164) for V6.0-000.
Revision V6.0-000	19 October 2012	In MUPIP SET, added the description of "-INST_freeze_on_error" (page 155).

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 181).

For MUPIP commands pertaining to multisite database replication, refer to Chapter 7: "Database Replication" (page 230).

> ### 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 45).

> ### 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.

> ### Note
>
> You can perform read operations on a GT.M database residing on a read-only mounted filesystem. However, please note:
>
> - The filesystem must remain read-only for the duration of any process that opens a database file resident on it. If a read-only file system is switched to read-write while GT.M processes have database files open on it, and other processes update those databases, the read-only processes are likely to read incorrect or corrupt data.
>
> - When the filesystem is read-only the shared memory resources which are typically shared among multiple processes instead become private to each process, so memory resource use increases with each additional concurrent process.
>
> - M locks mapped to regions that map to database files on read-only filesystems are visible only to the process that owns the locks, and are invisible to other processes.

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.

For information on the MUPIP commands and/or qualifiers that require standalone access, refer to "MUPIP Command Summary" (page 170).

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}]
 -BY[TESTREAM] [-NET[TIMEOUT]]
 -DA[TABASE]
 -[NO]NEWJNLFILES[=[NO]PREVLINK],[NO]S[YNC_IO]]
 -[NO]O[NLINE]
 -REC[ORD]
 -REPLA[CE]
 -RET[RY]=count
 -REPLIN[STANCE]=target_location
 -SH[OWPROGRESS]
 -SI[NCE]={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, disk mirroring, 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.

- 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 case insensitive and separated by a comma. 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 due to wildcards results ordering determined by M (ASCII) collation. MUPIP BACKUP selects regions for backup based on the order specified in the region-list.

- 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 specific 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.

- If you are backing up an encrypted datababase, you need to backup your $gtmcrypt_config file and securely preserve the encryption key. You also need to backup the related algorithms used to generate that key. For example, if you are using gpg for encryption and signing services, you must backup the relevant files in your $GNUPGHOME directory as part of your backup in addition to preserving the encryption key. Look in the manpages of gpg (man gpg) to see what all files require backup. Remember that if you forget the password of the encryption key which was used at the time of taking the backup, there is no way to decrypt the data from the encrypted regions. Therefore, ensure that you have procedures in place to securely handle password storage and retrieval of the encryption key.

- 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.

- When doing a complete backup, switch journal files as part of the backup command using the -NEWJNLFILES qualifier. This aligns the journal files with the backup and simplifies journal file retention. Use the NOPREVLINK option for this qualifier with caution if the original database is used for replication. If the link to the previous generation journal file is cut, then the source server cannot supply transactions from the prior generation journal files.

- 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 -DATABASE) 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 may be 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 turns 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 is in the backup database.

- Only one of the qualifiers DISABLE or OFF can be specified at any given point.

-Bytestream

Ttransfers 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 bytream 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:

```
-BY[TESTREAM]
```

- -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.

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 either sent over a TCP/IP connection or restored using the RESTORE command. The format of the NETTIMEOUT qualifier is:

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

• The default value is 30 seconds.

• Use only with: -BYTESTREAM and RESTORE.

-NEWJNLFILES

Determines the journaling charactertistics 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. Since only one new journal file is produced which may be used by both the original database and the backup copy, this option can cause NOPREVLINK errors with replication if the source server needs to supply transactions from a previous generation journal file.

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. In addition to storing the transaction number, -RECORD also stores a timestamp including the date and time of a successful backup operation in the database file. MUPIP DUMPFHEAD displays this timestamp as "sgmnt_data.last_start_backup" using a format such as "Sun Aug 15 13:37:32 2021". DSE DUMP -FILEHEADER -ALL also displays this date and time as "Last Record Backup Start" using the same format. These timestamps are displayed relative to the current timezone, and display as "Never" if the database file has never been successfully backed up with -RECORD. 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[ECORD]
```

- Use -RECORD (with the hypen) to timpestamp a refererence point and use RECORD as a keyword (as in -SINCE=RECORD) to specific 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 source server for the instance must be started at least once before backing up the replication instance file.

The format of the REPLINSTANCE qualifier is:

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

-RETry

Makes the specified number of retry attempts to perform BACKUP if the backup fails. If -RETRY is not specified, MUPIP retries file copy operation once if the first attempt fails for a retriable reason. In case of an error, retry attempts are always based on cp or pax even when another copy mechanism is available. If -RETRY=0 is specified, there is no retry for any failure that BACKUP encounters.

The format of the RETRY qualifier is:

```
-RET[RY]=<count>
```

-SHowprogress

When the backup copy mechanism supports monitoring, -SHOWPROGRESS periodically displays the transfer progress, estimated time left, speed, and the number of transaction applied to the database during backup.

The format of the SHOWPROGRESS qualifier is:

```
-SH[OWPROGRESS]
```

Incompatible with: -BYTESTREAM

-Since

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

```
-SI[NCE]={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[RANSACTION]=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 0x00000000000F42C3 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
MUPIP 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] [-[no]V6] [-R[EGION]=region-name]
```

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

Note that a GT.M logical database (an M global variable namespace) can consist of an arbitrary number of database files.

-V6

Directs MUPIP to create a V6.3-014 compatible database file. This option is negatable to ensure that MUPIP CREATE always creates current version database files even when the environment variable gtm_db_create_ver is defined as V6. By default, MUPIP CREATE creates database files for the current version. The environment variable gtm_db_create_ver can override the default behavior.

The format of the V6 qualifier is:

```
-[no]V6
```

-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
```

> **Note**
>
> MUPIP CREATE silently ignores requests to create database files that have AutoDB enabled in the Global Directory. MUPIP CREATE prints no message even if the database region is specified by name. For more information on AutoDB, refer to "Region Qualifiers" (page 71).

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|V63000A} file-name
```

> **Note**
>
> You must perform a GT.M database integrity check using the -noonline parameter prior to downgrading a database. The integrity check verifies and clears database header fields required for an orderly downgrade. If an integrity check is not possible due to time constraints, please rely on a rolling upgrade scheme using replication and / or take a backup prior to upgrading the database.

-VERSION={V4|V5|V63000A}

V4 and 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.

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.

V63000A:

V63000A downgrades to any release older than V6.3-001. All releases up to V6.3-000A have an empty 512-bytes block logically after the last usable database block. VERSION=V63000A changes this terminating block from having the current size of a database block to 512-bytes.

Examples for MUPIP DOWNGRADE

Example:

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

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

DUMPFHEAD

The MUPIP DUMPFHEAD command displays information about one or more database files. The format of the MUPIP DUMPFHEAD command is:

```
DU[MPFHEAD]  {-FI[LE] file-name |-REG[ION] region-list}
```

> ### Note
>
> MUPIP DUMPFHEAD obtains the field values from the database file header and ^%PEEKBYNAME obtains field values from the shared memory. Therefore, it may be possible that MUPIP DUMPFHEAD reports stale values for unflushed fields. If you require more recent file header values, use the MUPIP DUMPFHEAD -FLUSH which flushes file headers fields from shared memory to disk before returning the file header values.

-FILE=file

Specifies the name of the database file for the MUPIP DUMPFHEAD operation. -FILE does not require a Global Directory. The format of the FILE qualifier is:

```
-FI[LE] database-file-name
```

- The database file name must include the absolute or relative path.

- The -FILE qualifier is incompatible with the -REGION qualifier.

-FLUSH

Flushes the database file header flush before displaying the database file header fields. While this qualifier ensures the information is recent, under most production circumstances the output is sufficiently accurate without -FLUSH. Use of -FLUSH makes the operation significantly more heavyweight. If the database file header flush fails, MUPIP DUMPFHEAD -FLUSH produces a BUFFLUFAILED warning, but still produces output. The format of the FLUSH qualifier is:

```
-FL[USH]
```

-REGION=region

Specifies that the DUMPFHEAD command identifies one or more regions rather than a database file. The format of the REGION qualifier is:

```
-R[EGION] region-list
```

- The region-list identifies the target of DUMPFHEAD; region-list may specify more than one region of the current global directory in a list. DUMPFHEAD treats regions names (separated by commas) as case-insensitive with optional wildcard characters * and ? expanded in M (ASCII) collation order. Remember to escape wildcard characters to protect them from inappropriate expansion by the shell. DUMPFHEAD processes regions based on the order specified in the region-list.

- The region-list argument may specify more than one region of the current Global Directory in a list separated with commas. DUMPFHEAD -REGION requires the environment variable gtmgbldir to specify a valid Global Directory. For more information on defining gtmgbldir, refer to Chapter 4: "Global Directory Editor" (page 45).

- The -REGION qualifier is incompatible with the -FILE qualifier.

Examples for MUPIP DUMPFHEAD

Example:

```
$ mupip dumpfhead -file mumps.dat
```

This command lists information about the database file mumps.dat in the current working directory.

```
$ mupip dumpfhead -region "*"
```

This command lists information about all the database files mapped by the global directory specified by $gtmgbldir.

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 338).

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 MUPIP EXTRACT command is:

```
EXTR[ACT]
[
 -FO[RMAT]={GO|B[INARY]|Z[WR]}
 -FR[EEZE]
 -LA[BEL]=text
 -[NO]L[OG]
 -R[EGION]=region-list
 -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.

- 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.

- 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.

- The GO format is not supported for UTF-8 mode. Use BINARY or ZWR formats with UTF-8 mode.

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
1-2	Hexadecimal representation of the length of the label (by default 64 - decimal 100).
3-28	Fixed-length ASCII text containing: • "GDS BINARY EXTRACT LEVEL 6": when no region is encrypted. • "GDS BINARY EXTRACT LEVEL 8": when one more regions are encrypted using null IVs. • "GDS BINARY EXTRACT LEVEL 9": when one or regions are encrypted using non-null IVs.
29-41	Fixed-length ASCII text: Date and time of extract in the $ZDATE() format: "YEARMMDD2460SS".
42-48	Fixed-length ASCII text: Decimal maximum block size of the union of each region from which data was extracted.
49-55	Fixed-length ASCII text: Decimal maximum record size of the union of each region from which data was extracted.
56-62	Fixed-length ASCII text:Decimal maximum key size of the union of each region from which data was extracted.
63-69	Fixed-length ASCII text:Boolean indicator of Standard NULL collation (1) or GT.M legacy collation (0).
70-100	Fixed-length ASCII text: Space-padded label specified by the -LABEL qualifier; the default LABEL is "GT.M MUPIP EXTRACT"

BINARY Format Data File Header Label	
	For extracts in UTF-8 mode, GT.M prefixes UTF-8 and a space to -LABEL.

2. GO - Global Output format, used for files to transport or archive. -FORMAT=GO stores the data in record pairs. Each global node produces two records - the first contains the key and the second contains the value. MUPIP EXTRACT -FORMAT=GO has two header records - the first is a text label (refer to the LABEL qualifier) and the second is the date and time of extract in $ZDATE() format DD-MON-YEAR 24:60:SS. If -LABEL is not specified, the default first header is "GT.M MUPIP EXTRACT".

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 value. 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.

-Null_iv

Creates an encrypted binary extract with null IVs from a database with non-null IVs, which can be restored to a version that does not support non-null IVs. The format of the -NULL_IV qualifier is:

```
-[NO]NULL_IV
```

- GT.M versions prior to V6.3-000 used empty (all zeros or "NULL_IV") initialization vectors(IVs) to encrypt or decrypt -FORMAT="BINARY" extracts.

- GT.M version starting from V6.3-000 use non-zero IVs.

- Use the NULL_IV qualifier only on encrypted databases to create an encrypted binary extract in GDS BINARY EXTRACT LEVEL 8 format. This format can load data on any encrypted GT.M database created with a version prior to V6.3-000.

- The default is -NONULL_IV which produces a binary extract in GDS BINARY EXTRACT LEVEL 9 format.

-Region

Restricts MUPIP EXTRACT to a set of regions. The format of the REGION qualifier is:

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

region-list may specify more than one region of the current global directory in a list. Regions are case-insensitive, 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.

-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:

- 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.

- A suffix with a global name. For example, PIGEON* selects all global names from ^PIGEON through ^PIGEONzzzzz. You can use suffixes with a global name or a list.

> **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
```

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 after ensuring a consistent state between memory and secondary storage, which, with -ACCESS_METHOD=BG, means after flushing global buffers. 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 [[-ONL[INE] [-[NO]AUTORELEASE]] | [-NOONL[INE]] [-R[ECORD]]]}
▸ region-list
```

▼

- The region-list identifies the target of the FREEZE. region-list may specify more than one region of the current global directory in a list. Regions are case-insensitive, 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.

- MUPIP FREEZE waits for up to 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 unfreezes any regions it had previously marked as frozen and terminates with an error.

- 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.

- A MUPIP FREEZE -ON can specify either -NOONLINE, the default, or -ONLINE, and if -ONLINE, can specify either -AUTORELEASE, the default, or -NOAUTORELEASE.

- A FREEZE specifying -ONLINE attempts to minimize the impact of the FREEZE on concurrently updating processes by restricting all database-related updates to memory as long as it can.

- A FREEZE specifying -ONLINE -AUTORELEASE allows updates to continue immediately when GT.M needs to update the database file.

- After MUPIP FREEZE -ON -NOONLINE, 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.

- MUPIP FREEZE sends a DBFREEZEON/DBFREEZEOFF message to the system log for each region whose freeze state is changed.

- 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:

```
-[NO]A[UTORELEASE] - only valid with -ONLINE
-ON[LINE] - only valid with -ON
-OV[ERRIDE]
-R[ECORD] - only valid with -ON
```

-OFf

Clears a freeze set by another process with the same userid.

The format of the OFF qualifier is:

```
OF[F]
```

- A FREEZE -OFF which turns off a FREEZE -ONLINE -AUTORELEASE produces a OFRZNOTHELD warning to indicate that the freeze was automatically released and therefore did not protect whatever concurrent actions it was intended to guard.

- 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

-[NO]A[UTORELEASE

Controls the behavior of a FREEZE specified with -ONLINE when GT.M must write to a database file. The format of the AUTORELEASE qualifier is:

```
-[NO]A[UTORELEASE]
```

- -AUTORELEASE, the default, causes GT.M to release the freeze if it needs to update the file before a FREEZE -OFF.

- When FREEZE -ONLINE is in effect, MUPIP rejects all SET actions and produces the OFRZACTIVE warning; if you need to perform any such SET actions, you must release the freeze first.

- -NOAUTORELEASE causes GT.M to hold off actions that need to update the database file until someone issues a MUPIP FREEZE -OFF.

- The actions that require GT.M to write to the database file are:

  - Insufficient global buffers to hold updates - GT.M must flush buffers to make space to do any additional updates

- Insufficient space in the database to hold updates - GT.M must extend the file

- The journal file reaches its maximum size

- An epoch comes due - GT.M must create a checkpoint

- Someone issues a MUPIP BACKUP command - GT.M must record state information to mark the beginning of the backup

- When an -AUTORELEASE abandons a FREEZE, any actions that depend on the stability of the database file on secondary storage, such as a database copy, lose that protection and are not reliable, so they likely need to be repeated at a time when an -AUTORELEASE is less likely or when -NOONLINE is more appropriate.

- An -AUTORELEASE action produces an OFRZAUTOREL message in the operator log.

- An -AUTORELEASE action requires a FREEZE -OFF to reestablish a normal database state.

- Incompatible with: -OFF, -NOONLINE

-ONLine

Controls the potential impact of a FREEZE on concurrently updating processes. The format of the ONLINE qualifier is:

```
-[NO]ONL[INE]
```

- ON -NOONLINE, the default, causes the freeze to last until OFF, and makes management of the FREEZE straightforward.

- ON -ONLINE, causes GT.M to attempt to minimize the impact of the FREEZE on concurrently updating processes by taking a number of actions, as appropriate:

  - Switching journal files to provide maximum space

  - Performing an epoch to provide maximum time to the next epoch

  - Flushing the global buffers to make all available to hold updates

  - Incompatible with: -AUTORELEASE, -OFF

- After performing these preparations, -ONLINE allows updating processes to make updates to global buffers but defers flushing them to the database file.

- -ONLINE cannot apply to MM databases, so a FREEZE -ONLINE skips any MM regions it encounters.

- Refer to -AUTORELEASE above for additional information.

- Incompatible with: -OFF

> **Note**
>
> If a database is nearly full, and you about to use MUPIP FREEZE -ON -ONLINE, you may want to use MUPIP EXTEND first as a database file extension will either AUTORELEASE or "harden" the -ONLINE freeze effectively into a -NOONLINE freeze.

-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: -AUTORELEASE, -ON, -ONLINE -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.

- Incompatiable 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/appl/*.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-list or the journal/receiver pool in a table form.

The format of the MUPIP FTOK command is:

```
FT[OK] [-DB] [-JNLPOOL] [-RECVPOOL] [-ID] [-ONLY] [-[NO]HEADER] file-list
```

- file-list a space delimited list of files, such as that provided by the use of the * and ? shell wildcard characters.

- With -JNLPOOL or -RECVPOOL, MUPIP FTOK ignores any files in the list.

-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.

-ID

Specifies the signature use for the ftok; if unspecified, it defaults to the signature for a GT.M database file.

-ONLY

Restricts the output to only the FTOK Key; if a file is not valid and accessible, FTOK reports -1 values. -ONLY does not report the table header even when -HEADER is specified. If the database file is unavailable, the utility defaults to the -ONLY behavior.

-HEADER

Displays or omits the header of the table. By default, MUPIP FTOK displays the header.

Example:

```
$ mupip ftok -id mumps.dat
```

```
           File ::             Semaphore Id ::         Shared Memory Id ::              FTOK Key ::

           FileId

▶
▶  -------------------------------------------------------------------------------------------------------------
          mumps.dat :: 1044382209 [0x3e400201] ::  613417274 [0x2490013a] ::  725039842 [0x2b373ae2] ::
▶  0x00000000000e09818000002900000001

$ mupip ftok -id -only mumps.dat
          mumps.dat  ::   725039842  [ 0x2b373ae2 ]
$ mupip ftok -id -noheader mumps.dat
          mumps.dat :: 1044382209 [0x3e400201] ::  613417274 [0x2490013a] ::  725039842 [0x2b373ae2] ::
▶  0x00000000000e09818000002900000001

$ mupip ftok -jnlpool
            File ::             Semaphore Id ::         Shared Memory Id ::              FTOK Key ::

           FileId

▶
▶  -------------------------------------------------------------------------------------------------------------
          multi.repl ::             -1 [0xffffffff] ::          -1 [0xffffffff] ::  743218009 [0x2c4c9b59] ::
▶  0x00000000000e09838000002900000001
            jnlpool ::  636486196 [0x25f00234] ::  578814256 [0x22800130]
```

▼

HASH

Uses a 128 bit hash based on the MurmurHash3 algorithm to provide provides the hash of source files from the command line.

The format of the MUPIP HASH command is:

```
MUPIP HASH <file-names>
```

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 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 MUPIP INTEG command is:

```
I[NTEG]
[
 -A[DJACENCY]=integer
 -BL[OCK]=hexa;block-number
 -BR[IEF]
 -FA[ST]
 -FU[LL]
 -[NO]K[EYRANGES]
 -[NO]MAP[=integer]
 -[NO]MAXK[EYSIZE][=integer]
 -[NO]O[NLINE]
 -S[UBSCRIPT]=subscript]
 -TN[_RESET]
 -[NO]TR[ANSACTION][=integer]
]
{[-FILE] file-name|-REG[ION] region-list}
```

- 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 signs 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 preexisting database integrity issues.

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 400).

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 400). 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 accelerating database damage.

- By default, INTEG checks all active index and data blocks in the database.

- -FAST reports include adjacency information.

- 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 all "incorrectly marked free" errors as they require prompt action. MAP does not restrict their reports.

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_snaptmpdir can be used to indicate a directory where MUPIP should place the snapshot files (used by MUPIP INTEG -ONLINE). If gtm_snaptmpdir does not exist, INTEG uses the location specified by gtm_baktmpdir and if neither of those environment variables is defined, INTEG 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 wide variety of conditions.

- Temporary directory security settings must allow write access by the MUPIP process and 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.

- 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 command identifies one or more regions rather than a database file. The format of the REGION qualifier is:

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

- The region-list identifies the target of INTEG; region-list may specify more than one region of the current global directory in a list. INTEG treats regions names (separated by commas) as case-insensitive with optional wildcard characters * and ? expanded in M (ASCII) collation order. Remember to escape wildcard characters to protect them from inappropriate expansion by the shell. INTEG processes regions based on the order specified in the region-list.

- 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 Chapter 4: "Global Directory Editor" (page 45).

- 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:

```
-SU[BSCRIPT]=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

-Stats

Specifies INTEG to check any active statistics database associated with the region(s) specified for the command. The format of the STATS qualifier is:

```
-[NO]ST[ATS]
```

Specify STATS only if you have reason to understand that statistics reporting is failing with database errors or reporting incorrect results. Because -FILE requires standalone access and statistic databases are automatically created and removed it is incompatible with -STATS. The default is NOSTATS.

Incompatible with: -BLOCK, -FILE, -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             110           25.732            NA
Index             1170          341639           88.298             6
Data            340578          519489           99.268        337888
Free              6809              NA              NA            NA
Total           348559          861238              NA        337894
[Spanning Nodes:3329 ; Blocks:341403]
```

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 may be able to improve the performance of a database if the all blocks are as adjacent as possible. "% Used" is the amount of space occupied across the in-use blocks divided by the space available in the in-use blocks, and thus represents the packing density for the in-use blocks (excluding local bit maps). Higher "% Used" may actually be undesirable from a performance perspective as they indicate a higher likelihood of block splits with upcoming updates.

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             110           25.732            NA
Index             1170          341639           88.298             4
Data            340578          519489           99.268        337617
Free              6809              NA              NA            NA
Total           348559          861238              NA        337621
[Spanning Nodes:3329 ; Blocks:341403]
```

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
Directory          2               110           25.732         NA
Index           1170            341639           88.298          4
Data          340578               NA              NA         337617
Free            6809               NA              NA            NA
Total         348559               NA              NA         337621
```

Note the NA entries 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             0.585         NA
    0               1             109            50.878         NA
Global variable ^#t
Level           Blocks          Records         % Used      Adjacent
    1               1               1             0.585          0
    0               1              80            49.609          1
Global variable ^versionContent
Level           Blocks          Records         % Used      Adjacent
    1               1               1             0.585          0
    0               1               1            94.018          0
Global variable ^x
Level           Blocks          Records         % Used      Adjacent
    1               1               2             1.464          0
    0               2             109            52.551          1
```

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 181).

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]]}
-[O]NERROR={STOP|PROCEED|INTERACTIVE}
-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, which may be a UNIX file on any device that supports such files.

- LOAD accepts files with DOS style termination.

- 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. If the format of the input file is not specified, MUPIP LOAD automatically detects file format (BINARY/ZWR/GO) based on the file header of the input file. If format is specified, it must match the actual format of the input file for LOAD to proceed.

The format codes are:

```
B[INARY] - Binary format
GO - Global Output format
```

```
Z[WR] - ZWRITE format
```

- MUPIP LOAD detects file format (BINARY/ZWR/GO) based on the file header of extract files from MUPIP EXTRACT, ^%GO and DSE.

- -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).

- -FORMAT={ZWR|GO} applies to text files produced by tools such as MUPIP EXTRACT or %GO.

- For FORMAT={ZWR|GO} UTF-8 files not produced by MUPIP EXTRACT or %GO, the first line of the label must contain the case insensitive text "UTF-8".

- For all -FORMAT={ZWR|GO} files not produced by MUPIP EXTRACT or %GO, the second line should contain the case insensitive test "ZWR" for zwr format or "GLO" for GO format and the two label lines must contain a total of more than 10 characters.

- -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.

-BEgin

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.

-Onerror

Determines the MUPIP LOAD behavior when it encounters an error. The format of the ONERROR qualifier is:

```
-[O]NERROR={STOP|PROCEED|INTERACTIVE}
```

- STOP causes MUPIP LOAD to exit immediately.

- PROCEED proceeds to the next record.

- INTERACTIVE prompts to continue or stop.

By default MUPIP LOAD exits on encountering an error.

-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.

RCTLDUMP

Reports information related to relinkctl files and their associated shared memory segments. The format of the MUPIP RCTLDUMP command is:

```
MUPIP RCTLDUMP [dir1]
```

▼

If the optional parameter dir1 is not specified, MUPIP RCTLDUMP dumps information on all its active auto-relink-enabled directories (those with with a *-suffix) identified by $gtmroutines. With a directory path specified for dir1, MUPIP RCTLDUMP reports information on the one directory. An example output follows. It lists the full path of the Object directory; its corresponding relinkctl file name; the number of routines currently loaded in this relinkctl file; the number of processes including the reporting MUPIP process that have this Relinkctl file open; the shared memory id and length of the Relinkctl shared memory segment; one or more Rtnobj shared memory segment(s); and a listing of all the routine names loaded in this file (lines starting with rec#...).

- The Rtnobj shared memory line : All the length fields are displayed in hexadecimal. shmlen is the length of the allocated shared memory segment in bytes. shmused is the length that is currently used. shmfree is the length available for use. objlen

is the total length of all the objects currently loaded in this shared memory. As GT.M allocates blocks of memory with sizes rounded-up to an integer power of two bytes, shmused is always greater than objlen; for example with an objlen of 0x1c0, the shmused is 0x200.

- Lines of the form rec#... indicate the record number in the relinkctl file. Each relinkctl file can store a maximum of 1,000,000 records, i.e., the maximum number of routines in a directory with auto-relink enabled is one million. Each record stores a routine name (rtnname:), the current cycle for this object file record entry (cycle:) which gets bumped on every ZLINK or ZRUPDATE command, the hash of the object file last loaded for this routine name (objhash:), the number of different versions of object files loaded in the Rtnobj shared memory segments with this routine name (numvers:), the total byte-length of the one or more versions of object files currently loaded with this routine name (objlen:), the total length used up in shared memory for these object files where GT.M allocates each object file a rounded-up perfect 2-power block of memory (shmlen:).

Given a relinkctl file name, one can find the corresponding directory path using the Unix "strings" command on the Relinkctl file. For example, "strings /tmp/gtm-relinkctl-f0938d18ab001a7ef09c2bfba946f002", corresponding to the above MUPIP RCTLDUMP output example, would output "/obj" the corresponding directory name.

Example:

```
$ mupip rctldump .
Object Directory         : /tmp
Relinkctl filename       : /tmp/fis-gtm/V6.2-001_x86_64/gtm-relinkctl-61f9eb418212a24a75327f53106c1656
# of routines            : 1
# of attached processes  : 2
Relinkctl shared memory  : shmid: 11534344 shmlen: 0x57c6000
Rtnobj shared memory # 1 : shmid: 11567113 shmlen: 0x100000 shmused: 0x200 shmfree: 0xffe00 objlen: 0x1c0
 rec#1: rtnname: abcd cycle: 1 objhash: 0xedbfac8c7f7ca357 numvers: 1 objlen: 0x1c0 shmlen: 0x200
```

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.

MUPIP REORG can also encrypt a database and/or change the encryption keys for database files "on the fly" while the database is in use.

The format of the MUPIP REORG command is:

```
REO[RG]
[
 -D[OWNGRADE]
 -ENCR[YPT]=key
 -E[XCLUDE]=global-name-list
 -FI[LL_FACTOR]=integer
 -I[NDEX_FILL_FACTOR]=integer
 -K[EEP]={pct% | blocks}
 -NOCO[ALESCE]
 -NOSP[LIT]
 -NOSW[AP]
 -R[ESUME]
 -S[ELECT]=global-name-list
 -T[RUNCATE][=percentage]
 -UP[GRADE]
```

```
 -REG[ION] region-list
]
```

<table>
<tr>
<td>📌</td>
<td>

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.

</td>
</tr>
</table>

- As REORG is IO intensive, running a REORG concurrently with normal database access may impact the operation of normal processes. As the GT.M database engine has a daemonless architecture, attempts to reduce the impact by reducing the priority of REORG can (perhaps counter-intuitively) exacerbate rather than alleviate the impact. To reduce the impact REORG has on other processes, use the gtm_poollimit environment variable to limit the number of global buffers used by 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.

- Use MUPIP STOP (or <Ctrl-C> for an interactive REORG) to terminate a REORG process. Unless terminated with a kill -9, a REORG terminated by operator action or error is incomplete but does not adversely affect the database.

<table>
<tr>
<td>⚠️</td>
<td>

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.

</td>
</tr>
</table>

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:

-Downgrade

Downgrades blocks to a prior major version DB format. Use REORG -DOWNGRADE only when you have determined there is a clear reason to downgrade the DB format of a database to a prior format and there are no conditions that preclude that possibility.

-Encrypt

Encrypts an unencrypted database or changes the encryption key of a database while the database continues to be used by applications. Whether or not the prior encryption uses non-zero initialization vectors (IVs), database blocks encrypted with the new key use non-zero IVs The format of the ENCRYPT qualifier is:

```
-ENCR[YPT]=<key>
```

MUPIP provides <key> to the encryption plugin. The reference implementation of the plugin expects a key with the specified name in the encryption configuration file identified by $gtmcrypt_config. The configuration file must contain an entry in the database section for each database file mapping to a region specified in <region-list> that names the specified key as its key. The -ENCRYPT flag is incompatible with all other command line flags of MUPIP REORG except -REGION, and performs no operation other than changing the encryption key. If the specified key is already the encryption key of a database region,

MUPIP REORG -ENCRYPT moves on to the next region after displaying a message (on stderr, where MUPIP operations send their output).

As MUPIP REORG -ENCRYPT reads, re-encrypts, and writes every in-use block in each database file, its operations take a material amount of time on the databases of typical applications, and furthermore add an additional IO load to the system on which it runs. You can use the environment variable gtm_poollimit to ameliorate, but not eliminate, the impact, at the cost of extending execution times. To minimize impact on production instances, FIS recommends running this operation on replicating secondary instances, rather than on originating primary instances.

-ENCRYPT switches the journal file for each database region when it begins operating on it, and again when it completes, and also records messages in the syslog for both events.

As is the case under normal operation when MUPIP REORG -ENCRYPT is not active, journaled databases are protected against system crashes when MUPIP REORG -ENCRYPT is in operation: MUPIP JOURNAL -ROLLBACK / -RECOVER recovers journaled database regions (databases that use NOBEFORE journaling continue to require -RECOVER / -ROLLBACK - FORWARD).

Because a database file utilizes two keys while MUPIP REORG -ENCRYPT is underway, the database section of the configuration file provides for a single database file entry to specify multiple keys. For example, if the keys of database regions CUST and HIST, mapping to database files cust.dat and hist.dat in directory /var/myApp/prod, are to be changed from key1 to key2 using the command:

```
MUPIP REORG -ENCRYPT=key2 -REGION CUST,HIST
```

then the database section of the configuration file must at least have the following entries:

```
database: {
    keys: ({
        dat: "/var/myApp/cust.dat";
        key: "key1";
    },{
        dat: "/var/myApp/cust.dat";
        key: "key2";
    },{
        dat: "/var/myApp/hist.dat";
        key: "key1";
    },{
        dat: "/var/myApp/hist.dat";
        key: "key2";
    })
};
```

In other words, each database file entry can have multiple keys, and a key can be associated with multiple database files. With a configuration file that has multiple keys associated with the same database file, MUPIP CREATE uses the last entry. Other database operations use whichever key associated with the database file has a hash matching one in the database file header, reporting an error if no key matches. To improve efficiency when opening databases, you can delete entries for keys that are no longer used from the configuration file.

MUPIP REORG -ENCR[YPT] can encrypt an unencrypted database only if the following command:

```
    MUPIP SET -ENCRYPTABLE -REGION <region-list>
```

has previously marked the database "encryptable".

The command requires standalone access to the database. It performs some basic encryption setup checks and requires the gtm_passwd environment variable to be defined and the GNUPGHOME environment variable to point to a valid directory in the environment. Just as encrypted databases use global buffers in pairs (for encrypted and unencrypted versions of blocks), a database marked as encryptable has global buffers allocated in pairs (that is, the actual number of global buffers is twice the number reported by DSE DUMP -FILEHEADER) and requires correspondingly larger shared memory segments. To revert unencrypted but encryptable databases back to "unencryptable" state, use the command:

```
MUPIP SET -NOENCRYPTABLE -REGION <region-list>
```

The above command only requires standalone access, and the result depends on the state of the database. It:

- is a no-op if the database is encrypted;

- is disallowed if the database is partially (re)encrypted; and

- prohibits encryption if the database is not encrypted.

Under normal operation, a database file has only one key. Upon starting a MUPIP REORG -ENCRYPT to change the key, there are two keys, both of whose hashes GT.M stores in the database file header. With a MUPIP REORG -ENCRYPT operation underway to change the key, normal database operations can continue, except for another MUPIP REORG -ENCRYPT or MUPIP EXTRACT in binary format. Other MUPIP operations, such as MUPIP BACKUP and MUPIP EXTRACT in ZWR format can occur. A MUPIP REORG -ENCRYPT operation can resume after an interruption, either unintentional, such as after a system crash and recovery, or intentional, that is, an explicit MUPIP STOP of the MUPIP REORG -ENCRYPT process. To resume the REORG operation, reissue the original command, including the key parameter. (Note that supplying a key other than the one used in the original command produces an error.)

After the MUPIP REORG -ENCRYPT process completes, subsequent MUPIP REORG -ENCRYPT operations on the same region(s) are disallowed until the following command is run:

```
MUPIP SET -ENCRYPTIONCOMPLETE -REGION <region-list>
```

Blocking subsequent MUPIP REORG -ENCRYPT operations after one completes provides time for a backup of the entire database before enabling further key changes. MUPIP SET -ENCRYPTIONCOMPLETE reports an error for any database region for which MUPIP REORG -ENCRYPT has not completed.

> ### Note
>
> - FIS recommends rotating (changing) the encryption key of the database for better security. The frequency of encryption key rotation depends on your security requirements and policies.
>
> - MUPIP REORG -ENCRYPT does not enable switching between encryption algorithms. 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 230) for more complete documentation.

-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
```

- 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[NDEX_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.

-Keep

For a MUPIP REORG -TRUNCATE, -KEEP allows the user to specify, as a decimal integer number of blocks or a percentage from 0% to 99% of the starting total blocks, the total amount of space the truncate operation should leave (not truncate) in the database file at the completion of other REORG operations. The format of the KEEP qualifier is:

```
-KEEP={pct%|blocks}
```

- Where pct is a decimal integer percentage followed by a percent-sign (%) and alternatively blocks is a decimal integer number of blocks. If REORG cannot meet the keep value, it produces an informational message indicating that it did not perform any truncation, and how many free blocks it left.

-NOCOalesce

-NOCOALESCE specifies to MUPIP REORG to skip actions that increase block density. The format of the NOCOALESCE qualifier is:

```
-NOCO[alesce]
```

- By default, MUPIP REORG attempts to rearrange blocks which are below the specified fill-factor to pack data more densely. More tightly packed data can decrease block I/O and reduce the storage requirements.

-NOSPlit

-NOSPLIT specifies to MUPIP REORG to skip actions that decrease block density. The format of the NOSPLIT qualifier is:

```
-NOSP[lit]
```

- By default, MUPIP REORG attempts to rearrange blocks which are above the specified fill-factor to pack data less densely. Having empty space in blocks can reduce the need for block splits going forward, which may improve performance.

-NOSWap

-NOSWAP specifies to MUPIP REORG to skip actions that increase block adjacency. The format of the NOSWAP qualifier is:

```
-NOSW[ap]
```

- By default, MUPIP REORG attempts to rearrange blocks so logically related blocks have adjacent block numbers. On rotating storage, this tends to improve performance; on solid state storage the results are device specific and may increase device wear. Swap activities tend to generate a lot of journal file volume.

-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[ESUME]
```

- 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
```

region-list may specify more than one region of the current global directory in a list. Regions are case-insensitive, 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.

-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.

> **Warning**
>
> Because any competing updates may interfere with MUPIP's ability to free space, minimize or prevent competing updates.

-UPgrade

Upgrade blocks to current DB version format. Use REORG -UPGRADE when it is a part of a release notes procedure to upgrade the DB version format of a database or when you encounter a database integrity error that requires running REORG -UPGRADE.

Examples for MUPIP REORG

Example:

```
$ mupip reorg
Fill Factor:: Index blocks 100%: Data blocks 100%

Global: CUST (region DEFAULT)
Blocks processed    : 667340
```

```
Blocks coalesced    : 601487
Blocks split        : 0
Blocks swapped      : 319211
Blocks freed        : 646964
Blocks reused       : 298814
Blocks extended     : 0

Global: HIST (region HIST)
%GTM-I-FILERENAME, File /var/myApp/prod/journals/hist.mjl is renamed to /var/myApp/prod/journals/
hist.mjl_2015289165050
Blocks processed    : 337069
Blocks coalesced    : 12888
Blocks split        : 0
Blocks swapped      : 329410
Blocks freed        : 315998
Blocks reused       : 308337
Levels Eliminated   : 1
Blocks extended     : 0
$
```

In this output:

- Blocks processed - the number of blocks originally used by the global variable

- Blocks coalesced - the number of blocks that were sufficiently compacted enough to free a block

- Blocks split - the number of blocks expanded enough to require the allocation of a new block

- Blocks swapped - the number of blocks moved to improve adjacency; this can exceed the number of blocks processed as a consequence of the movement of blocks

- Blocks freed - the number of blocks formerly used that were released by a combination of swaps and coalesces

- Blocks reused - blocks freed, and then reused

- Levels eliminated - reduction in the depth of the global variable tree

- Blocks extended - the number of blocks the database grew during the reorg

Note also that owing the database update activity of REORG, the hist.mjl journal file reached its limit, requiring MUPIP to switch the journal file.

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 months + 1 month "safety" margin) * five percent per month). The REORG command based on the above assumptions is as follows:

```
$ mupip reorg -fill_factor=80
```

Example:

The following example uses recorg -encrypt to encrypt a database "on the fly". This is a simple example created for demonstration purposes. It is NOT recommended for production use. Consult your GT.M support channel for specific instructions on encrypting an unencrypted database.

Create an empty default unencrypted database.

```
$gtm_dist/mumps -r ^GDE exit
  $gtm_dist/mupip create
```

Setup the GNUPG home directory.

```
export GNUPGHOME=$PWD/.helengnupg3
mkdir $GNUPGHOME # Ensure that you protect this directory with appropriate permissions.
chmod go-rwx $GNUPGHOME
```

Create a new key. Enter demo values. This example uses demo values from the database encryption example in the GT.M Acculturation Workshop document. Accept default values. Choose a strong passphrase.

```
gpg --gen-key
```

Edit the key to add a new sub-key:

```
gpg --edit-key helen.keymaster@gt.m
```

Type addkey, select option 6 RSA (encrypt only), and accept default values and execute the following commands:

```
gpg --gen-random 2 32 | gpg --encrypt --default-recipient-self --sign --armor > gtm_workshop_key.txt
gpg --decrypt < ./gtm_workshop_key.txt | gpg --encrypt --armor --default-recipient-self --output
▶ gtm.key
```


Refer to the 'man gpg; a description on the qualifiers for gpg.

Create a gtmcrypt_config file as following:

```
$ cat config
database: {
    keys:   (
               {
               dat: "/path/to/mumps.dat" ;
               key: "/path/to/gtm.key" ;
               }
            );
  }
```

Set the environment variable gtmcrypt_config to point to this config file.

```
export gtmcrypt_config=$PWD/config
```

Set the enviroment varible gtm_passwd.

```
echo -n "Enter passphrase for gtm.key: " ; export gtm_passwd=`$gtm_dist/plugin/gtmcrypt/maskpass|cut -f 3 -d "
▶ "`
```

▼

Execute the following commands:

```
$ mupip set -encryptable -region DEFAULT
$ mupip reorg -encrypt="gtm.key" -region DEFAULT
mupip reorg -encrypt="gtm.key" -region DEFAULT
Region DEFAULT : MUPIP REORG ENCRYPT started
Region DEFAULT : Database is now FULLY ENCRYPTED with the following key: gtm.key
Region DEFAULT : MUPIP REORG ENCRYPT finished
```

Execute the following command when encryption completes.

```
$ mupip set -encryptioncomplete -region DEFAULT
Database file /home/gtc_twinata/staff/nitin/tr11/mumps.dat now has encryption marked complete
```

Always keep the keys in a secured location. Always set gtmcrypt_config and gtm_passwd to access the encrypted database.

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 230) .

RESTORE

Incrementally restores the output of the BACKUP -BYTESTREAM to a database file at the backup location.

The format of the RESTORE command is:

```
RE[STORE] [-NET[TIMEOUT]] [-[NO]E[XTEND]] dbfile bytestream-bkup-list|<TCP_socket_address>
```

- **dbfile** identifies the name of the database file that RESTORE uses as a starting point.

- When BACKUP -BYTESTREAM directs the output to disks or pipes, **bytestream-bkup-list** specifies a comma (,) seperated list of one or more bytestream files to RESTORE into the database file specified with file-name. The list and must be in sequential order, from the oldest transaction number to the most recent transaction number. **bytestream-bkup-list**may also include a list of pipe commands.

  <TCP_socket_address> specifies a combination of IPv4 or IPV6 hostname and a port number (for example, tcp:// hostname1:portno) for which RESTORE sets up a listening process to restore the bytestream data from an active BACKUP - BYTESTREAM process to the database file. In both these cases:

  - The transaction number in the first incremental backup must be one more than the current transaction number of the database. Otherwise, the integrity checking mechanism of MUPIP RESTORE produces an error.

  - You can incrementally restore BACKUP -BYTESTREAM data from either a TCP stream or multiple bytestream files to only one database file with one MUPIP RESTORE command.

- If the BACKUP -BYTESTREAM 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.

- For more information on the -NETTIMEOUT qualifier, refer to "-NETtimeout" (page 101) in the MUPIP BACKUP section of this chapter.

-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.

```
$ mupip backup -bytestream region1,region2 tcp://hostname1:port1,tcp://hostname1:port2
▶ -nettimeout=420
```

▼

This command starts a MUPIP process waiting for a listening MUPIP RESTORE processes to send bytestream data of region1 and region2.

```
$ mupip restore region1.dat tcp://hostname1:port1 -nettimeout=420
```

This command start a TCP listening process to receive the bytstream data from region1 of the previous MUPIP BACKUP -BYTSTREAM command.

```
$ mupip restore region2.dat tcp://hostname1:port2 -nettimeout=420
```

This command start a TCP listening process to receive the bytstream data from region2 of the previous MUPIP BACKUP -BYTSTREAM command. Note that you cannot start RESTORE for region2 before the backup of region1 completes.

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 to 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 MUPIP RUNDOWN command is:

```
RU[NDOWN] {-FILE file-name|-REGION region-list|-RELINKCTL [dir]|-OVERRIDE}
```

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.

A successful MUPIP RUNDOWN of a database region removes any current MUPIP FREEZE.

RUNDOWN -FILE can be directed to a statistics database file and works even if the corresponding actual database file does not exist.

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[ile]
-R[egion]=region-list
-RELinkctl [dir1]
-Override
```

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. MUPIP RUNDOWN with no argument removes any statistics database file resources associated with actual database file resources it can remove.

-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

-Override

Overrides the protection that prevents MUPIP 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.

-Region

The region-list identifies the target of the RUNDOWN. region-list may specify more than one region of the current global directory in a list. Regions are case-insensitive, 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.

Use the wildcard "*" to rundown all inactive regions in a global directory.

Incompatible with: -FILE

When MUPIP 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.

-Relinkctl

Cleans up orphaned Relinkctl files. FIS strongly recommends avoiding actions that tend to make such cleanup necessary - for example, kill -9 of GT.M processes or ipcrm -m of active Relinkctl and/or Rtnobj shared memory segments.

If the optional dir1 is not specified, MUPIP RUNDOWN -RELINKCTL examines the environment variable $gtmroutines, attempts to verify and correct their attach counts and runs down all its inactive auto-relink-enabled directories (those with with a *-suffix). Alternatively, one can specify a directory path for the parameter dir1 and MUPIP RUNDOWN -RELINKCTL treats it as an auto-relink-enabled directory and runs down the resources associated with this one directory. It prints a RLNKCTLRNDWNSUC message on a successful rundown and a RLNKCTLRNDWNFL message on a failure (usually because live processes are still accessing the Relinkctl file).

SEMAPHORE

Reports the details of a space delimited list of semaphores IDs.

The format of the MUPIP SEMAPHORE command is:

```
MUPIP SEMAPHORE <sem-ids-list>
```

SET

Use MUPIP SET for performance tuning and/or modifying certain database and journal file attributes.

The format of the SET command is:

```
SE[T] {-FI[LE] file-name|-JN[LFILE] journal-file-name|-REG[ION] region-list}
 -AC[CESS_METHOD]={BG|MM}
 -[NO]AS[YNCIO]
 -[NO]DE[FER_TIME][=seconds]
 -[NO]DEFER_ALLOCATE
 -[NO]ENCRYPTA[BLE]
 -ENCRYPTI[ONCOMPELTE]
 -EX[TENSION_COUNT]=integer(no of blocks)
```

```
-F[LUSH_TIME]=integer
-FU[LLBLKWRT]={0|1|2}
-G[LOBAL_BUFFERS]=integer
-H[ARD_SPIN_COUNT]=integer
-[NO]INST[_FREEZE_ON_ERROR]
-JN[LFILE] journal-file-name journal-file-quals
-K[EY_SIZE]=bytes
-[NO]LCK_SHARES_DB_CRIT
-L[OCK_SPACE]=integer
-M[UTEX_SLOTS]=integer
-N[ULL_SUBSCRIPTS]=value
-[NO]Q[DBRUNDOWN]
-[NO]REA[D_ONLY]
-REC[ORD_SIZE]=bytes
-REG[ION] region-list
-REP[LICATION]={ON|OFF}
-RES[ERVED_BYTES]=integer]
-SL[EEP_SPIN_COUNT]=integer
-SPIN_SLEEP_M[ASK]=hex_mask
-[NO]STAT[S]
-STATSD[B_ALLOCATION]=integer(no of blocks)
-[NO]STD[NULLCOLL]
-T[RIGGER_FLUSH]=integer
-V[ERSION]={V4|V6}
-W[AIT_DISK]=integer
-WR[ITES_PER_FLUSH=integer
```

- The file-name, journal_file_name, region-list or -REPLICATION qualifier identify the target of the SET.

- The SET command must include one of the following target qualifiers which determine whether the argument to the SET is a file-name or a region-list.

-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: -JNLFILE, -REGION and -REPLICATION

-Jnlfile

Specifies that the argument is a journal-file-name. The format of the JNLFILE qualifier is:

```
-JNLF[ILE] journal-file-name
```

Incompatible with: -FILE, -REGION and -REPLICATION

-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] region-list
```

The region-list identifies the target of SET. region-list may specify more than one region of the current global directory in a list. Regions are case-insensitive, separated by a comma, and wild-cards can be used to specify them. Any region-name may include the wild-card 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.

Incompatible with: -FILE, -JNLFILE and -REPLICATION

-REPlication

Specifies whether replication is on or off. The format of the REPLICATION qualifier is:

```
-REP[LICATION]={ON|OFF}
```

Incompatible with: -JNLFILE

The following sections describe the action qualifiers of the MUPIP SET command exclusive of the details related to journaling and replication, which are described in Chapter 6: "GT.M Journaling" (page 181) and Chapter 7: "Database Replication" (page 230). All of these qualifiers are incompatible with the -JNLFILE and -REPLICATION qualifiers.

-ACcess_method

Specifies the access method (GT.M buffering strategy) for storing and retrieving data from the database file. The format of the ACCESS_METHOD qualifier is:

```
-AC[CESS_METHOD]=code
```

For more information on specifying the ACCESS_METHOD, refer to "Segment Qualifiers" (page 76).

-ASyncIO

Specifies whether to use asynchronous I/O for an access method BG database, rather than using synchronous I/O through the file system cache. ASYNCIO is incompatible with the MM access method and an attempt to combine the two with MUPIP SET produces a ASYNCIONOMM error. The format of the ASYNCIO qualifier is:

```
-[NO]AS[YNCIO]
```

For more information on specifying ASYNCIO,refer to "Segment Qualifiers" (page 76).

-Defer_time

Specifies, in MM access mode, the multiplying factor applied to the flush time to produce a wait after an update before ensuring a journal buffer write to disk; the default is 1. A value of 2 produces a wait of double the flush time. -NODEFER_TIME or a value of -1 turns off timed journal writing, leaving the journal, under light update conditions, to potentially get as stale as the epoch time. Note that, in MM mode without the sync_io option set, absent a VIEW("JNLFLUSH") from the application, GT.M only fsyncs the journal at the epoch. The format of the DEFER_TIME qualifier is:

```
-[NO]D[efer_time][=seconds]
```

-DEFER_Allocate

With -DEFER_ALLOCATE, GT.M instructs the file system to create the database file as a sparse file. Before using -DEFER_ALLOCATE, ensure that your underlying file system supports sparse files. By default UNIX file systems, and GT.M, use

sparse (or lazy) allocation, which defers actual allocation until blocks are first written. The format of the DEFER_ALLOCATE qualifier is:

```
-[NO]DEFER_Allocate
```

- Utilities such as du report typically show lower disk space usage for a database file with -DEFER_ALLOCATE because GT.M instructs the file system to defer disk space allocation to the time when there is an actual need. With -NODEFER_ALLOCATE, such utilities report higher disk space usage count as GT.M instructs the file system to preallocate disk space without waiting for a need to arise.

- -DEFER_ALLOCATE makes database file extensions lighter weight. However, disk activity may tend towards causing fragmentation.

- To switch an existing database file so it immediately preallocates all blocks, first use MUPIP SET -NODEFER_ALLOCATE to set the switch in the database file header, followed by **MUPIP EXTEND -BLOCKS=n**, where **n >= 0**. Failures to preallocate space produce a PREALLOCATEFAIL error.

- The default is DEFER_ALLOCATE.

-Epochtaper

Tries to minimize epoch duration by reducing the number of buffers to flush by GT.M and the file system (via an fsync()) as the epoch (time-based or due to journal file auto-switch) approaches. Epoch tapering reduces the impact of I/O activity during an epoch event. Application that experience high load and/or need to reduce latency may benefit from epoch tapering. The format of the -EPOCHTAPER qualifier is:

```
-[NO]EPOCHTAPER
```

-ENCRYPTAble

Performs some basic encryption checks and marks the database as encryptable. Note that marking a database as encryptable does not encrypt the database.For more information on encrypting a database, refer to "-Encrypt" (page 138). The format of the ENCRYPTABLE qualifier is:

```
-[NO]ENCRYPTA[BLE]
```

-ENCRYPTIoncomplete

Checks whether a prior MUPIP REORG -ENCRYPT completed successfully and marks encryption as complete. For usage scenarios of MUPIP SET -ENCRYPTIONCOMPLETE, refer to "-Encrypt" (page 138). The format of the ENCRYPTIONCOMPLETE qualifier is:

```
-ENCRYPTI[ONCOMPELTE]
```

-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:

```
-EX[TENSION_COUNT]=integer
```

For more information on specifying the EXTENSION_COUNT, refer to "Segment Qualifiers" (page 76).

-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
```

-FUllblkwrt

Determines whether GT.M writes only valid database blocks contents, or a full block including meaningless trailing content. Full block writes are more efficient with some secondary storage because they avoid read-before-write. The format of the FULLBLKWRT qualifier is:

```
-FU[LLBLKWRT]={0|1|2}
```

- When -FULLBLKWRT=2, a process writes all newly allocated database blocks in their entirety regardless of their actual valid contents. This relieves some file systems from tracking as much unallocated space and thus reduces file system metadata maintenance.

- When -FULLBLKWRT=1, a process writes entire file system blocks in their entirety regardless of their actual valid contents, on some file systems, this avoids reading in advance of most writes and thus reduces file system load and increases response time.

- When -FULLBLKWRT=0 (the default), a process writes only valid data. FIS advises using the same value of -FULLBLKWRT for all processes.

Note that when the file system block size and the database block size are the same there is no difference between the settings of 1 and 2.

-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 "Segment Qualifiers" (page 76).

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 on memory constrained systems 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 2,097,151 buffers, but may vary depending on your platform. 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.

-Hard_spin_count

Specifies the maximum amount of time for processes to sleep while waiting to obtain critical sections for shared resources, principally those involving databases. The format of the -SPIN_SLEEP_MASK qualifier is:

```
-HARD_SPIN_COUNT=integer
```

The mutex hard spin count specifies the number of attempts to grab the mutex lock before initiating a less CPU-intensive wait period. The format of -HARD_SPIN_COUNT is:

• The default value is 128

• Except on the advice of your GT.M support channel, FIS recommends leaving the default values unchanged in production environments, until and unless, you have data from testing and benchmarking that demonstrates a benefit from a change.

-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. The format of the INST_FREEZE_ON_ERROR qualifier is:

```
-[NO]INST[_FREEZE_ON_ERROR]
```

For more information on creating a list of custom errors that automatically cause an Instance Freeze, refer to "Instance Freeze" (page 254).

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 297) section of the Database Replication chapter.

-Journal

Specifies whether the database allows journaling and, if it does, characteristics for the journal file.:

> **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 JOURNAL qualifier is:

```
-[NO]J[OURNAL][=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 "SET -JOURNAL Options " (page 197).

-Key_size

Specifies the maximum key size in bytes for storing and retrieving data from the global database file. The maximum supported size is 1019 bytes. The format of the KEY_SIZE qualifier is:

```
-K[EY_SIZE=bytes
```

For more information on KEY_SIZE, refer to "Region Qualifiers" (page 71).

-Lck_shares_db_crit

Specifies whether LOCK actions share the same resource and management as the database or use a separate resource and management. The format of the LCK_SHARES_DB_CRIT qualifier is:

```
-[NO]LC[K_SHARES_DB_CRIT]
```

The default is Sep(arate)/FALSE.

For more information, refer to the -[NO]L[OCK_CRIT] section of GDE "Region Qualifiers" (page 71).

-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 262144 pages.

- The minimum LOCK_SPACE is 10 pages.

- The default LOCK_SPACE is 40 pages.

- For more information on LOCK_SPACE, refer to "Segment Qualifiers" (page 76).

- This qualifier requires standalone access.

-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. The format of the MUTEX_SLOTS qualifier is:

```
-M[UTEX_SLOTS]=integer
```

-Null_subscripts

Controls whether GT.M accepts null subscripts in database keys.

Usage:

```
-N[ULL_SUBSCRIPTS]=value
```

- 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.

-PROblksplit

Specifies the approximate number of records a block in the region must exceed before GT.M considers a proactive block split. If the specified value is 0, GT.M does not engage in proactive block splitting. The format of the -PROBLKSPLIT qualifier is:

```
-PRO[BLKSPLIT]=integer
```

- GT.M makes proactive block splits when a block has a history of causing repeated restarts; GT.M never proactively splits blocks that don't encounter restarts.

- Proactive block splits tend to make globals require more blocks and potentially deeper trees.

- A value of one (1) indicates to always consider proactive splits, which might be useful for regions used to hold globals with a first subscript of, for example, $job.

- The default value is five (5).

-Qdbrundown

Shortens normal process shutdown when a large number of processes accessing a database file need to shutdown almost simultaneously, for example, in benchmarking scenarios or emergencies. The format of the QDBRUNDOWN qualifier is:

```
-[NO]Q[DBRUNDOWN]
```

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 that the last process to exit might leave the database shared memory and IPC resources in need of cleanup. Except after the number of concurrent processes exceeds 32Ki, QDBRUNDOWN minimizes the possibility of abandoned resources, but it cannot eliminate it. When using QDBRUNDOWN, use an explicit MUPIP command such as RUNDOWN or JOURNAL -RECOVER or -ROLLBACK of the database file after the last process exits, to ensure the cleanup of database shared memory and IPC resources; not doing so risk database damage.

When a database has QDBRUNDOWN enabled, if the number of attached processes ever exceeds 32Ki, GT.M stops tracking the number of attached processes, which means that it cannot recognize when the number reaches zero (0) and the shared resources can be released. The process that detects this event issues a NOMORESEMCNT in the system log. This means an orderly, safe shutdown requires a MUPIP JOURNAL -ROLLBACK -BACKWARD for replicated databases, a MUPIP JOURNAL -RECOVER -BACKWARD for unreplicated journaled databases and a MUPIP RUNDOWN for journal-free databases.

-PArtial_recov_bypass

Sets the CORRUPT_FILE flag in the database file header to FALSE. The CORRUPT_FILE flag indicates whether a region completed a successful recovery. The format of the PARTIAL_RECOV_BYPASS qualifier is:

```
-PA[RTIAL_RECOV_BYPASS]
```

For more information, refer to the CORRUPT_FILE qualifier in "CHANGE -FIleheader Qualifiers" (page 364).

-Read_only

Indicates whether GT.M should treat an MM access method segment as read only for all users, including root. This designation augments UNIX authorizations and prevents any state updates that normally might require an operational action for a database with no current accessing (attached) processes. Setting the -READ_ONLY attribute on a database file with BG access method or setting the access method to BG on a -READ_ONLY database produces an error. The GT.M help databases have -READ_ONLY set by default. The format of the READ_ONLY qualifier is:

```
-[NO]REA[D_ONLY]
```

-NOREAD_ONLY makes a database file writable for users who have write permissions on the database file. Note that because of the way GT.M handles semaphores for READ_ONLY database files it does not enforce standalone access for setting the characteristic from READ_ONLY to NOREAD_ONLY. However, making a change from READ_ONLY to NOREAD_ONLY without standalone access will likely cause problems, such as errors on termination and failure to release IPC resources that would require additional MUPIP commands to clean up. Therefore FIS recommends using other means such as the following based on fuser return of one (1) to verify there are no processes are accessing the file:

```
fuser <db-filename> | awk -F: '{if(length($NF))exit(1)}' && $gtm_dist/mupip set -noread_only -file
 <db-filename>
```

> **Note**
>
> When the first process connects to a database, it creates a access-control semaphore as part of management of the shared resource. However, when a process connects to a -READ_ONLY database, each create a private copy of the in-memory structures for the database and thus a private semaphore.

-RECord_size

Specifies the maximum record size in bytes for storing and retrieving data from the global database file. The maximum supported size is 1MiB bytes. The format of the RECORD_SIZE qualifier is:

```
-REC[ORD_SIZE=bytes
```

For more information on KEY_SIZE, refer to "Region Qualifiers" (page 71).

-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 = Block Size - Key Size - 40 (Block/Record header overheads).

-SLeep_spin_count

Specifies the number of times a processes suspends its activity while waiting to obtain critical sections for shared resources, principally those involving databases. The format of the -SLEEP_SPIN_COUNT qualifier is:

```
-SLEEP_SPIN_COUNT=integer
```

- **integer** is the number times the process yields to the OS scheduler or sleeps (depending in the SPIN_SLEEP_LIMIT) after exhausting its hard spin count and before enquing itself to be awakened by another process releasing the shared resource mutex.

- The default is 128.

- Except on the advice of your GT.M support channel, FIS recommends leaving the default values unchanged in production environments, until and unless, you have data from testing and benchmarking that demonstrates a benefit from a change.

-Spin_sleep_mask

Specifies the maximum amount of time for processes to sleep while waiting to obtain critical sections for shared resources, principally those involving databases. The format of the -SPIN_SLEEP_MASK qualifier is:

```
-SPIN_SLEEP_MASK=hex_mask
```

- **hex_mask** is a hexidecimal mask that controls the maximum time (in nanoseconds) the process sleeps on a sleep spin.

- The default is zero (0) which causes the process to return control to the UNIX kernel to be rescheduled with no explicit delay. When the value is non-zero, the process waits for a random value between zero (0) and the maximum value permitted by the mask.

- Except on the advice of your GT.M support channel, FIS recommends leaving the default values unchanged in production environments, until and unless, you have data from testing and benchmarking that demonstrates a benefit from a change.

-STATs

Specifies whether GT.M should permit statistics sharing for this region. This characteristic permits operational exclusion of statistics sharing for a region. The format of the STATS qualifier is:

```
-[NO]STAT[S]
```

- At database creation, GDE controls this characteristic, which, by default it specifies as STATS (on). When on, this characteristic causes GT.M to create a small MM database for the associated region to hold the shared statistics.

- A process disables itself from maintaining the shared statistics when it fails to open a statsDB. It does not, however, disable subsequently starting processes from maintaining the shared statistics.

-StatsDB_allocation

Specifies the initial number of GDS blocks for the shadow stats regions when the database region is first created. This qualifier requires standalone access and a file or region name. The format of the STATSDB_ALLOCATION qualifier is:

```
-STATSD[B_ALLOCATION]=integer
```

- GT.M doubles the block count with each statsdb size increase.

- The minimum STATSDB_ALLOCATION is 128 blocks.

- The maximum STATSDB_ALLOCATION is 8,388,607 blocks.

- By default, GDE uses an STATSDB_ALLOCATION of 2050 blocks.

-STDnullcoll

Specifies whether GT.M uses standard MUMPS collation or GT.M collation for null-subscripted keys. FIS strongly recommends that you use STDNULLCOLL and against using this non-standard null collation, which is the default for historical reasons. The format of the STDNULLCOLL qualifier is:

```
-[NO]STD[NULLCOLL]
```

-Trigger_flush

Specifies the decimal value, in buffers, for the threshold at which processes start flushing dirty buffers after each update. The format of the TRIGGER_FLUSH qualifier is:

```
-T[RIGGER_FLUSH]=integer
```

-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(s).

-WAit_disk

Specifies the seconds to wait for disk space before giving up on a database block write, where zero (0) means to give an error immediately without waiting. The format of the WAIT_DISK qualifier is:

```
-WA[IT_DISK]=seconds
```

-WRites_per_flush

Specifies the decimal number of blocks to write in each flush. The default value is 7. The format of the WRITES_PER_FLUSH qualifier is:

```
-WR[ITES_PER_FLUSH]=integer
```

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[euristic]=estimation_technique] [-se[lect]=global-name-list] [-r[egion]=region-list]
▶ [-a[djacency]=integer] [-su[bscript]]=global-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[euristic]={sc[an][,level=<lvl>] | a[rsample][,samples=<smpls>] |
  i[mpsample][,samples=<smpls>]}
```

▼

- smpls is the number of samples and must be greater than zero (0)

- lvl is a positive or negative tree level designation and -(level of the root block) <= lvl <= (level of the root block)

estimation-technique is one of the following:

- **scan,level=<lvl>**

  Traverses the global variable tree and counts the actual number of records at levels from the root down to the level specified by lvl; the number of records at a level is the same as the number of blocks at the next lower level. The default is 1, which gives the count of level 0 blocks, which are the blocks containing data rather than metadata. 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.

  Specifying, a level 0 to get a record count for data increases the processing work close to an order of magnitude.

- **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.

The 2 sigma column for the two sampling techniques addresses the dispersion of the samples (in blocks) by giving the probability (rounded to a whole percentage) that the actual value falls farther away from the reported value by more than two sigma; in other words, larger values indicate less confidence in the accuracy of the result. With the scan method the "sample" is "complete," so any inaccuracy comes from concurrent updates.

> ### ⚠️ 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.

-Adjacency=integer

Specifies the logical adjacency of data blocks that MUPIP SIZE should assume during estimation. By default, MUPIP SIZE assumes -ADJACENCY=10 and reports the logical adjacency in the "Adjacent" column of the MUPIP SIZE report. Note that adjacency is only a proxy for database organization and its usefulness may be limited by the technology and configuration of your secondary storage. See the INTEG section of this chapter for additional comments on adjacency.

-Select

Specifies the global variables on which MUPIP SIZE runs. -SELECT is incompatible with -SUBSCRIPT.If neither -SELECT or -SUBSCRIPT is specified, MUPIP SIZE selects all global variables in the specified region(s).

The format of the SELECT qualifier is:

```
-s[elect]=global-name-list
```

global-name-list can be:

- A comma separated list of global variables. For example, -select='^a,^x)'

- Global variables may be subscripted. For example, -select='^g1("""a""")'. Note that string subscripts require sufficient quotes to make it though the shell parsing to reach MUPIP.

- A range of global variables denoted by start:end syntax. For example, -select='^g1:^g4("""c""")'.

- A global variable with wildcards, for example, -select='^g*'. Note 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.

-Subscript

Specifies the subscripted/non-subscripted global variables on which MUPIP SIZE runs. -SUBSCRIPT is incompatible with -SELECT.

The format of the SUBSCRIPT qualifier is:

```
-su[bscript]=global-list
```

global-name-list can be:

- A range of global variables denoted by start:end syntax. For example, -subscript="^g1:^g4".

- A range of subscripted global variables denoted by start:end syntax. For example, -subscript="^g(1):^g(4)".

> **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.

> **Caution**
>
> On receipt of a MUPIP STOP signal, a GT.M process cleans up its participation in managing the database before shutting down. On receipt of three MUPIP STOP signals in a row, a GT.M process shuts down forthwith without cleaning up - the equivalent of a kill -9 signal. This can result in structural database damage, because GT.M does not have sufficient control of what happens in response to an immediate process termination to protect against database damage under all circumstances.
>
> In all cases, on receipt of a MUPIP STOP, a process will eventually terminate once it gets the resources needed to clean up. Use three MUPIP STOPs in a row only as a last resort, and when you do, perform a MUPIP INTEG at your earliest opportunity thereafter to check for database structural damage, and repair any damage following the procedures in Chapter 11 (Maintaining Database Integrity).
>
> You may never have to perform a MUPIP STOP if your application is designed in a way that it reduces or eliminates the probability of a process getting in the final try of a transaction. For more information, refer to the Programmers Guide.

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>]|-UPGRADE}
```

Before you run the MUPIP TRIGGER command:

1. Set the value of the environment variable **gtmgbldir**: 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 all planned trigger definitions. 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]]
```

- For information on the syntax and usage of a trigger definition file, refer to the Triggers chapter and the $ZTRIGGER() section in the Functions chapter of the **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. Trigger maintenance operations reserve their output until the transaction commits at which time they deliver the entire output in a consistent way. 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.

- MUPIP TRIGGER replicate trigger definitions as logical actions from an originating/primary instance to a replicating/secondary instance based on LGTRIG journal records. This permits the instances to have different sets of triggers and differing database layouts (for example, different # of regions, different block sizes, different maximum-record-size, and so on).

- MUPIP TRIGGER error messages associated with loading triggers limit trigger expression source lines to 80 characters including a trailing ellipsis to indicate there was more text, and they also replace any non-graphic characters with a dot (.)

- GT.M triggers apply to spanning regions. When $ZTRIGGER() or MUPIP TRIGGER define triggers that apply to globals spanning multiple regions, each of the spanned regions install a definition.

- 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-separate list of global variables or trigger names. The format of the SELECT qualifier is:

```
-SELE[CT][=name-list*][ <select-output-file>]
```

- 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.

- With no arguments specified, GT.M treats -SELECT as -SELECT="*" and extracts a list of all current triggers.

- 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.

- MUPIP TRIGGER -SELECT displays all output including errors on STDOUT.

- For Trigger definition reporting operations, $ZTRIGGER("SELECT") and MUPIP TRIGGER -SELECT, return a non-zero exit status when their selection criteria encounter an error in the select.

- MUPIP TRIGGER -SELECT works even if a multi-line XECUTE string does not terminate with a newline character. For more information on multi-line XECUTE strings, refer to the -xecute="|<<strlit1"|>> section under Trigger Definition File in the Triggers chapter and the $ZTRIGGER() section in the Functions chapter of the **GT.M Programmer's Guide**.
  in the Programmers Guide.

> **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 node.

> **Important**
>
> MUPIP TRIGGER treats the deletion of a non-existent trigger as a success; if that is the only operation, or one of a set of successful operations, it returns success 0 to the shell. Also, MUPIP TRIGGER returns failure in case of trigger selection using trigger names where the number after the pound-sign (#) starts with a 0 (which is an impossible auto-generated trigger name).

UPGRADE

Upgrades older trigger definitions into current format.

The format of the UPGRADE qualifier is:

```
-UPGRADE
```

If GT.M encounters an old trigger definition it produces a NEEDTRIGUPGRD message. To preserve the possibility of a straightforward downgrade to an earlier version, perform a select "*" action with MUPIP TRIGGER (or $ZTRIGGER() and save the result. Note that TRIGGER -UPGRADE assumes that the existing trigger definitions are properly defined; if the prior release has produced defective triggers delete them with a wild-card ("*"), and redefine the triggers in the new release. In the event of a downgrade, delete "*" all triggers before the downgrade and insert the saved version from before the upgrade. Attempting to perform a MUPIP TRIGGER -UPGRADE on a database without write authorization to the database produces a TRIGMODREGNOTRW error. The -UPGRADE qualifier is not compatible with any other MUPIP TRIGGER qualifier. Trigger

upgrades from older versions may produce journal records based on the prior format that a MUPIP JOURNAL -RECOVER cannot process correctly, therefore, FIS recommends you do them with journaling off, and start with a backup and fresh journal files after the trigger upgrade.

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"):

1. 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!"""
```

2. 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.

3. 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.

1. Begin by executing the following command:

```
$ mupip trigger -select="^Acct*"Output file:
```

2. 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!"""
```

3. 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!"""
```

4. Execute a command like the following:

```
$ mupip trigger -triggerfile=trigger_mod.trg
```

5. 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"):

1. Begin by executing the following command:

```
$ mupip trigger -select="^Acct*"Output file:
```

2. 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!"""
```

3. 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**.

4. 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 –xecute="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 –xecute="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	GT.M version*	Standalone Access#
B[ACKUP]	region-name file-name	-BK[UPDBJNL]=DISABLE | OFF		
		-BY[TESTREAM] -NET[TIMEOUT]=seconds		
		-C[OMPREHENSIVE]		
		-DA[TABASE] -REPLA[CE]		
		-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		Y
DO[WNGRADE]	file-name	-V[ERION]={V4|V5}		Y
DU[MPFHEAD]	file-name or region-list	-	V6.3-001	
EN[DIANCVT]	file-name	-OUTDB=<outdb-file>		Y
		-OV[ERRIDE]		Y
EXI[T]	-	-		N/A
EXTE[ND]	region-name	-B[LOCKS]=blocks		
EXTR[ACT]	-	-FO[RMAT]=GO|B[INARY]|Z[WR]		
		-FR[EEZE]		
		-NU[LL_IV]	V6.3-000	
		-LA[BEL]=text		
		-[NO]L[OG]		
		-S[ELECT]=global-name-list		

COMMAND	OBJECTS	MAIN QUALIFIER	GT.M version*	Standalone Access#
		-O[CHSET]=character-set		
		-R[EGION=region-list		
F[REEZE]	region-list	-OF[F] [-OV[ERRIDE]]		
		-ON [-[NO]ONL[INE] [-[NO]A[UTORELEASE]] [-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]		Y
		-FU[LL]		
		-NO]K[EYRANGES		
		-[NO][MAP]=integer		
		-[NO]MAXK[EYSIZE]=integer		
		-R[EGION]		
		-[NO]ST[ATS]		
		-SU[BSCRIPT]=subscript		
		-TN[_RESET]		Y
		-[NO]TR[ANSACTION][=integer]		
IN[TRPT]	process id			N/A
J[OURNAL]	file-name	-EX[TRACT][=file-specification|		
		-stdout]	V5.5-000	
		-REG[ION]	V6.2-000	
		-REC[OVER]	-RO[LLBACK]		
		-SH[OW][=show-option-list]		
		-[NO]V[ERIFY]		

COMMAND	OBJECTS	MAIN QUALIFIER	GT.M version*	Standalone Access#
		-BA[CKWARD] | -FO[RWARD]		
		-CO[RRUPTDB]	V6.3-010	
L[OAD]	file-name	-BE[GIN]=integer		
		-BLOCK_DENSITY		
		-E[ND]=integer		
		-FI[LLFACTOR]=integer		
		-FO[RMAT]=GO|B[INARY]|Z[WR]		
		-S[TDIN]	V5.5-000	
		-O[NERROR]		
REO[RG]		-D[OWNGRADE]		
		-ENCR[YPT]=key	V6.3-000	
		-E[XCLUDE]=global-name-list		
		-FI[LL_FACTOR]=integer		
		-I[NDEX_FILL_FACTOR]=integer		
		-K[EEP]={pct%|blocks}	V6.3-009	
		-NOCO[ALESCE]	V6.3-009	
		-NOSP[LIT]	V6.3-009	
		-NOSW[AP]	V6.3-009	
		-R[ESUME]		
		-S[ELECT]=global-name-list		
		-T[RUNCATE][=percentage]	V5.5-000	
		-UP[GRADE]		
		-REG[ION] region-list		
REP[LICATE]	file-name	-E[DITINSTANCE]		
		-N[AME]	V5.5-000	
		-I[NSTANCE_CREATE]		
		-R[ECEIVER]		
		-S[OURCE]		
		-UPDA[TEPROC]		
RE[STORE]	file-name or file-list	-[NO]E[XTEND]		Y

COMMAND	OBJECTS	MAIN QUALIFIER	GT.M version*	Standalone Access#
RU[NDOWN]	file-name or region-name	-F[ILE]		Y
		-R[EGION]		
		-RELINKCTL [dir]		
		-OVERRIDE		
SE[T]	file-name or region-name	SE[T] {-FI[LE] file-name|-JN[LFILE] journal-file-name|-REG[ION] region-list|-REP[LICATION]={ON|OFF}}		
		-AC[CESS_METHOD]={BG|MM}		Y
		-[NO]AS[YNCIO]		Y
		-[NO]DE[FER_TIME][=seconds]		Y
		-[NO]DEFER_ALLOCATE	V6.2-002	
		-[NO]EP[OCHTAPER]	V6.2-001	
		-[NO]ENCRYPTABLE	V6.3-000	Y
		-E[XTENSION_COUNT]=integer(no of blocks)		
		-F[LUSH_TIME]=integer		
		-FU[LLBLKWRT]={0|1|2}	V6.3-012	Y
		-G[LOBAL_BUFFERS]=integer		Y
		-H[ARD_SPIN_COUNT]=integer	V6.3-003	
		-[NO]INST[_FREEZE_ON_ERROR]	V6.0-000	
		-JN[LFILE] journal-file-name		Y
		-K[EY_SIZE]=bytes		Y
		-[NO]LCK_SHARES_DB_CRIT		Y
		-L[OCK_SPACE]=integer		Y
		-M[UTEX_SLOTS]=integer	V6.0-002	Y
		-N[ULL_SUBSCRIPTS]=value	V6.3-003	Y
		-PA[RTIAL_RECOV_BYPASS]		Y
		-[NO]Q[DBRUNDOWN]	V6.0-000	Y
		-[NO]REA[D_ONLY]	V6.3-003	Y
		-REC[ORD_SIZE]=bytes		Y
		-REG[ION] region-list		

COMMAND	OBJECTS	MAIN QUALIFIER	GT.M version*	Standalone Access#
		-REP[LICATION]={ON|OFF}		Y
		-RES[ERVED_BYTES]=integer		Y
		-SLEE[P_SPIN_COUNT]=integer		
		-SPIN[_SLEEP_MASK]=hexa_mask	V6.3-003	
		-STAN[DALONENOT]		Y
		-[NO]STD[NULLCOLL]	V6.3-003	Y
		-[NO]STAT[S]		Y
		-STATSD[B_ALLOCATION]=integer	V7.0-001	Y
		-V[ERSION]={V4|V6}		
		-W[AIT_DISK]=integer		Y
SI[ZE]	global-name-list region-list	-H[EURISTIC]=estimation_technique	V6.0-001	
		-S[ELECT]=global-name-list	V6.0-001	
		-R[EGION]=region-list	V6.0-001	
ST[OP]	process-id	process-id		N/A
TRIGGER	-	-TRIG[GERFILE]=<trigger_definitions_file>	V5.4-002	
		-NOPR[OMPT]	V5.4-002	
		-SELE[CT][=name-list|*][<select-output-file>]	V5.4-002	
		-UPGRADE	V5.4-002	
UP[GRADE]	file-name	-		Y

The following table summarizes the qualifiers.

Main Qualifier	MUPIP Command	Options/Qualifiers	GT.M Version*	Standalone Access#
-EDITINSTANCE	REPLICATE	-CHANGE		
		-DETAIL		
		-OFFSET=hexa		
		-VALUE=hexa		
		-SIZE=hexa		
		-VALUE=hexa		
		-[NO]QDBRUNDOWN	V6.3-000	

Main Qualifier	MUPIP Command	Options/Qualifiers	GT.M Version*	Standalone Access#
-FENCES=<fence-options-list>	JOURNAL-RECOVER-ROLLBACK	ALWAYS		
		NONE		
		PROCESS		
-OFF	FREEZE	-OVERRIDE		
		-RECORD		
-ON	FREEZE	-[NO]ONLINE		
		-[NO]AUTORELEASE		
-INSTANCE_CREATE	REPLICATE	-NAME		
		-NOREPLACE		
		-SUPPLEMENTARY	V5.5-000	
		-[NO]QDBRUNDOWN	V6.3-000	
-JOURNAL=<journal-options-list>	BACKUP SET	ALIGNSIZE=integer		
		ALLOCATION=integer		
		AUTOSWITCHLIMIT=integer		
		BEFORE_IMAGES		
		BUFFER_SIZE=integer		
		DISABLE		
		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		

Main Qualifier	MUPIP Command	Options/Qualifiers	GT.M Version*	Standalone Access#
-RECEIVER	REPLICATE	-AUTOROLLBACK	V5.5-000	
		-BUFFSIZE=integer		
		-CHANGELOG		
		-CHECKHEALTH		
		-CMPLVL=integer		
		-FILTER=filter_name		
		-he[lpers]=[m[,n]]		
		-INITIALIZE	V6.0-000	
		-CMPLVL=n		
		-LISTENPORT=integer [		
		-AUTOROLLBACK[=VERBOSE]]	V5.5-000	
		-LOG=logfile		
		-LOG_INTERVAL=integer		
		-NORESYNC	V5.5-000	
		-RESUME=strm_num	V5.5-000	
		-REUSE=instname	V5.5-000	
		-RSYNC_STRM=strm_num	V5.5-000	
		-SHOWBACKLOG		
		-SHUTDOWN		
		-START		
		-STATSLOG=[ON|OFF]		
		-STOPRECEIVERFILTER	V6.3-009	
		-STOPSOURCEFILTER		
		-TIMEOUT=seconds		
		-TLSID=label	V6.1-000	
		-UPDATEONLY		
		-UPDOK	V5.5-000	
		-UPDNOTOK	V5.5-000	
		-UPDATERESYNC=/path/to/bkup-orig-inst	V5.5-000	
-RECOVER	JOURNAL	-AFTER=time		

Main Qualifier	MUPIP Command	Options/Qualifiers	GT.M Version*	Standalone Access#
		-APPLY_AFTER_IMAGE		
		-BACKWARD		
		-BEFORE=time		
		-[NO]BROKENTRANS[=file]		
		-CHAIN		
		-CHECKTN		
		-[NO]ER[ROR_LIMIT][=integer]		
		-FENCES=fence-option-list		
		-FORWARD		
		-FULL		
		-GLOBAL=<global_list>		
		-ID=<pid_list>		
		-INTERACTIVE		
		-LOOKBACK_LIMIT=<lookback_limit_options>		
		-[NO]LOSTTRANS[=file]		
		-PARA[LLEL][=n]	V6.3-000	
		-RED[IRECT]=file-pair-list		
		-SINCE=time		
		-VERBOSE		
		-VERIFY		
-EXTRACT	JOURNAL	-AFTER=time		
		-BEFORE=time		
		-[NO]BROKENTRANS[=file]		
		-CHAIN		
		-CHECKTN		
		-[NO]ER[ROR_LIMIT]=integer]		
		-FENCES=fence-option-list		
		-FULL		
		-GLOBAL=<global_list>		
		-GV[PATFILE]=/path/to/pattern-file	V6.3-009	

Main Qualifier	MUPIP Command	Options/Qualifiers	GT.M Version*	Standalone Access#
		-ID=<pid_list>		
		-INTERACTIVE		
		-LOOKBACK_LIMIT=<lookback_limit_options>		
		-[NO]LOSTTRANS[=file]		
		-REGION		
		-SINCE=time		
		-VERBOSE		
		-VERIFY		
-ROLLBACK	JOURNAL	-APPLY_AFTER_IMAGE		
		-BACKWARD		
		-BEFORE=time		
		-[NO]BROKENTRANS[=file]		
		-[NO]ER[ROR_LIMIT][=integer]		
		-FENCES=fence-option-list		
		-FETCHRESYNC		
		-LOOKBACK_LIMIT=<lookback_limit_options>		
		-[NO]LOSTTRANS[=file]		
		-ONLINE	V5.5-000	
		-PARA[LLEL][=n]	V6.3-000	
		-RES[YNC]=hexa:journal_sequence_number		
		-VERBOSE		
		-VERIFY		
-SHOW=<show-option-list>	JOURNAL	-ACTIVE_PROCESSES		
		-ALL		
		-BROKEN_TRANSACTIONS		
		-HEADER		
		-PROCESSES		
		-STATISTICS		
		-AFTER=time		

Main Qualifier	MUPIP Command	Options/Qualifiers	GT.M Version*	Standalone Access#
		-USER=user-list		
		-TRANSACTION=[KILL|SET]		
		-INTERACTIVE		
		-GLOBAL=<global_list>		
		-ID=<pid_list>		
		-INTERACTIVE		
		-PARA[LLEL][=n]	V6.3-000	
-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		
		-FREEZE=on|off	V6.0-000	
		-[NO]COMMENT=string		
		-INSTSECONDARY=secondary_instance name		
		-NOJNLFILEONLY	V6.2-001	
		-JNLPOOL-LOG=log_file		
		-LOG_INTERVAL=integer		
		-LOSTTNCOMPLETE		
		-NEEDRESTART		

Main Qualifier	MUPIP Command	Options/Qualifiers	GT.M Version*	Standalone Access#
		-PASSIVE		
		-[NO]PLAINTEXTFALLBACK	V6.1-000	
		-PROPAGATEPRIMARY		
		-RENEGOTIATE_INTERVAL=minutes		
		-ROOTPRIMARY		
		-SECONDARY=secondary_instance_name		
		-SHOWBACKLOG		
		-SHUTDOWN		
		-START		
		-STATSLOG		
		-STOPSOURCEFILTER		
		-TIMEOUT=seconds		
		-TLSID=label	V6.1-000	
		-UPDOK		
		-UPDNOTOK		
		-ZEROBACKLOG		
-VERSION={V4|V5}	DOWNGRADE UPGRADE	file-name		

* denotes the GT.M version in which the qualifier or command was first introduced. This column is maintained for qualifiers and command introduced in GT.M versions starting with GT.M V5.4-002.

Additional information about Standalone Access:

- Although MUPIP EXTRACT does not require standalone access, without the -FREEZE qualifier to prevent the database to prevent updates, it does not guarantee a consistent state. Therefore, from an application standpoint, you might plan for a standalone access during a MUPIP EXTRACT operation.

- Although MUPIP LOAD does not require standalone 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.

- MUPIP commands that need standalone access issue a MUUSERLBK error on a crashed replication-enabled database and MUUSERECOV error in case of a non-replicated-but-journaled database.

Chapter 6. GT.MJournaling

Revision History		
Revision V7.0-002	29 March 2022	• In "JOURNAL " (page 204), removed a reference to working with journal files on tape
Revision V7.0-001	24 November 2021	• In "Journal Action Qualifiers" (page 207), add that file can be a fifo • In "Journal Files " (page 184), GTM-9443 MUPIP SET waits with the removal of a temporary journal file. • In "Introduction " (page 184), time of creation -> time of rename • In "MUPIP Command Summary" (page 170), The -STATSD[B_ALLOCATION] qualifier expects to get an integer.
Revision V7.0-000	12 February 2021	• In "MUPIP Command Summary" (page 170), removed -DBG as it is for internal FIS use.
Revision V6.3-013	30 June 2020	• In "Journal Time Qualifiers" (page 215), change "ending time" to "ending inclusive time" • In "Journal Extract Formats" (page 224), Pull the version specific simple extract header. Add examples of both simple and detail extracts. Add a description of the detail format with the record offsets and lengths. • In "MUPIP Command Summary" (page 170), add a new Standalone Access column; correct note about EXTRACT; change -B[YTESTREAM] to -BY[TESTREAM]; change STD[NULLSUBS] to STD[NULLCOLL]; remove SPIN[_SLEEP_LIMIT] • In "SET -JOURNAL Options " (page 197), Maximum journal buffer size
Revision V6.3-012	08 April 2020	• In "MUPIP Command Summary" (page 170), add -FULLBLKWRT
Revision V6.3-010	31 October 2019	• In "Journal Control Qualifiers" (page 218), add the CORRUPTDB qualifier. • In "MUPIP Command Summary" (page 170), add qualifiers for REORG; add -CORRUPTDB
Revision V6.3-009	27 June 2019	• In "Journal Selection Qualifiers" (page 221), add new GVPATFILE qualifier. • In "MUPIP Command Summary" (page 170), add entries for -GVPATFILE and -STOPRECEIVERFILTER.

Revision V6.3-007	04 February 2019	• In "Backup Journal Files " (page 190), Add caution on using NOPREVLINK
		• In "Journal Control Qualifiers" (page 218), add NO for BROKENTRANS and LOSTTRANS
		• In "JOURNAL " (page 204), add NO for LOSTTRANS and BROKENTRANS
		• In "Journal Sequence Number Qualifiers" (page 217), add links to the relevant sections.
		• In "MUPIP Command Summary" (page 170), add [NO] for LOSTTRANS and BROKENTRANS qualifiers
		• In "-RECover " (page 208), add statement on specifying non-overlapping file names
		• In "SET -JOURNAL Options " (page 197), add information about JNLSPACELOW and made general improvements
Revision V6.3-006	26 October 2018	• In "Backward Recovery " (page 187), remove an incorrect reference to -AFTER.
		• In "Broken Transaction File" (page 189), content improvements.
		• In "Epoch" (page 190), add new section.
		• In "Examples for MUPIP SET" (page 203), remove the NOPREVJNLFILE example as it is not supported as a JOURNAL option.
		• In "SET Object Identifying Qualifiers " (page 195), UTF-8 tweaks
		• In "Journal Time Qualifiers" (page 215), add more clarity to the description of the -AFTER qualifier.
		• In "Lost Transaction File" (page 189), content improvements
		• In "-RECover " (page 208), add a note stating that -RECOVER on replicated databases initiates database recovery but turns replication OFF.
		• In "SET -JOURNAL Options " (page 197), remove the PREVJNLFILE option and UTF-8 tweaks.
Revision V6.3-004	23 March 2018	• In "Journal Sequence Number Qualifiers" (page 217), add information about the -seqno qualifer of journal -extract.
		• In "MUPIP Command Summary" (page 170), remove the entry for -updhelper

Revision V6.3-003	12 December 2017	• In "MUPIP Command Summary" (page 170), add a new GT.M Version column - maintainable since GT.M V5.4-002.
		• In "-ROLLBACK [{-ON[LINE]|-NOO[NLINE]}]" (page 210), remove obsolete caution
Revision V6.3-002	22 August 2017	• In "SET -JOURNAL Options " (page 197), change default BUFFER_SIZE value to 2312; improve some wording
		• In "-SHow=show-option-list " (page 212), adjust sample MUPIP JOURNAL -SHOW output to match current format
Revision V6.3-001	20 March 2017	• In "Journal Action Qualifiers" (page 207), added information about the new PARALLEL qualifier.
		• In "Journal Control Qualifiers" (page 218), added for the -CHECKTN and FENCE qualifiers; clarified FENCES
		• In "JOURNAL " (page 204), made adjustments to the qualifier compatibility matrix for MUPIP JOURNAL. Expanded the qualifier entries to their full keywords.
		• In "-PARA[LLEL][=n]" (page 208), fixed typo; removed text associated with older versions and with other options
		• In "Journal Sequence Number Qualifiers" (page 217), used caps for non example keywords
		• In "Journal Time Qualifiers" (page 215), added changes for BEFORE and SINCE
		• In "-[NO]Verify" (page 214), added improvements and revised recommendations.
		• In "Journal Extract Formats" (page 224), removed HP-UX and OpenVMS
		• In "MUPIP Command Summary" (page 170), added FREEZE, DOWNGRADE, DUMPFHEAD commands and MUPIP INTEG and SET qualifiers for V6.3-001
		• In "-ROLLBACK [{-ON[LINE]|-NOO[NLINE]}]" (page 210), specified that -ONLINE is not supported for -ROLLBACK -FORWARD and made content improvements; reordered for better flow and removed some passive voice
		• In "SET -JOURNAL Options " (page 197), used delimiters for qualifiers
Revision V6.2-001	27 February 2015	In "Journal Extract Formats" (page 224), added information about the LGTRIG field.
Revision V6.0-001/1	22 March 1013	• Improved the formatting of all command syntaxes.

		• In "SET Object Identifying Qualifiers " (page 195), added information about the -repl_state and -dbfilename qualifiers.
Revision V6.0-001	27 February 2013	In "-EXtract[=<file-name>|-stdout]" (page 207), added information about the gtm_extract_nocol environment variable.
Revision V6.0-000/1	21 November 2012	Updated "SET -JOURNAL Options " (page 197) and the journaling limits for V6.0-000.

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 rename 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).
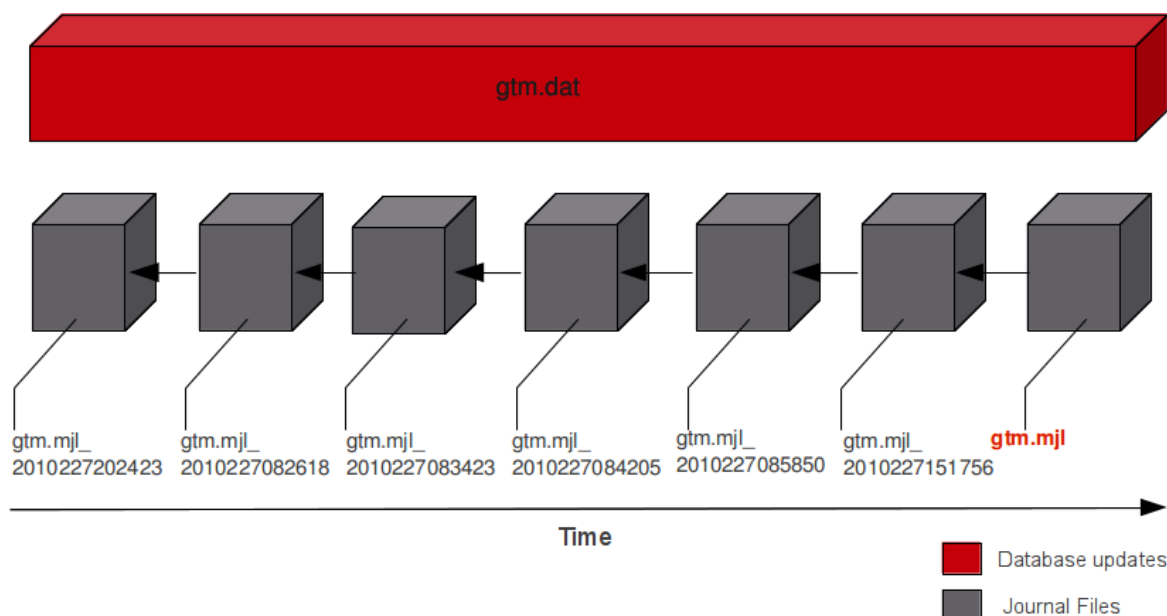
At any given time the database file (gtm.dat) has a single active journal file (gtm.mjl) with links to predecessor ("previous generation") journal files. The black arrow between the journal files demonstrate how a journal file is back-linked to its predecessor with file name in the form of gtm.mjl_YYYYJJJHHMMSS 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 using the timestamp of the time of switch. 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...]]" 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 creates 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 MUPIP process detects an .mjl_new file and no .mjl file, it deletes .mjl_new only after concluding that has been abandoned by seeing that the file persists for an interval longer than the renaming MUPIP process should take, 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 441).

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.

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) 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.
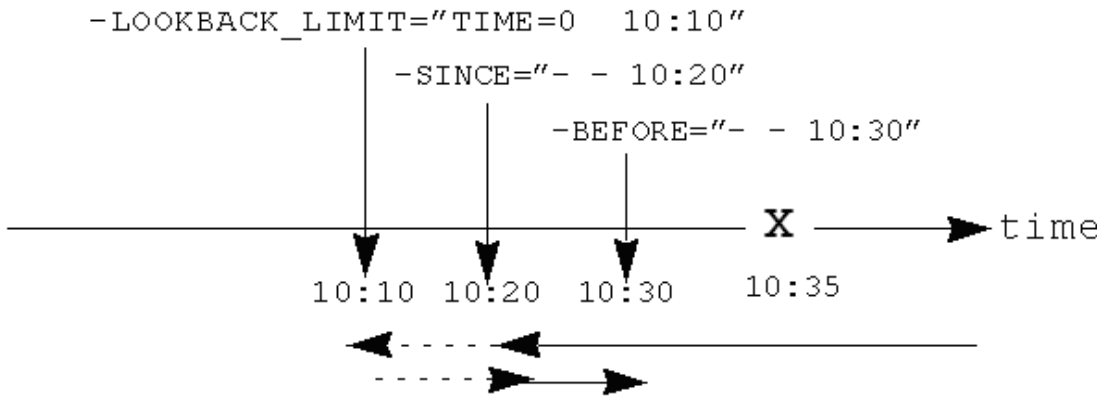
-LOOKBACK_LIMIT controls the maximum amount of additional backward processing, in this case, 10 minutes. Note that the -SINCE time in this example is slightly exaggerated for the sake of the graphical representation. If the application includes TSTART and TCOMMIT commands to fence transactions, backward 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 you 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 256).

Broken Transaction File

GT.M reports unfinished journal records and incomplete fenced transactions as broken transactions. Unfinished journal records appear when certain catastrophic events prevent GT.M from completely writing all journal records to the journal file. Incomplete fenced transactions appear when a missing journal file or the selection qualifiers used with MUPIP JOURNAL reduce the coverage of journal records in a way that only a portion of a fenced transaction becomes available for processing. For example, when database updates between the TSTART and TCOMMIT commands span to multiple regions and MUPIP JOURNAL -EXTRACT -FORWARD <jnlfile> attempt to process journal records from only one region of that transaction. MUPIP JOURNAL omits broken transaction records from processing and stores them into a file called the broken transaction file.

Lost Transaction File

Any complete transaction (fenced or not) that occurs after a broken transaction is a lost transaction. MUPIP JOURNAL omits lost transaction records from processing and stores them into a file called the lost transaction file. The label of the journal file provides information about the MUPIP JOURNAL processing (ROLLBACK, RECOVER, or EXTRACT) that generates the lost transaction file.

For -EXTRACT and -RECOVER, MUPIP JOURNAL processing generates a lost transaction file for all complete transactions (fenced or not) after a broken transaction. For -ROLLBACK, MUPIP JOURNAL generates a lost transaction file with records that may include unreplicated updates, in-flight updates, or updates that were rolled back due to an operator intervention or a replication setup reconfiguration. If you are in a multisite replication configuration, a lost transaction is a transaction that must be rolled off a database to maintain consistency across all instances.

As MUPIP JOURNAL omits processing the records in a lost transaction file, you need to reconcile (as needed) them to your application. If you are using replication, you should apply a lost transaction file to the originating instance as soon as possible.

Use the $ZQGBLMOD() function to help determine whether it is safe to apply a record from a lost transaction file to a global name. If you are not using replication, FIS recommends creating application tools/operation scripts that help with reprocessing the information in the lost transaction file as part of post-recovery check procedures.

Epoch

An epoch is a checkpoint at which GT.M creates a state where a database file and its journal file are in complete sync and to which GT.M can make a consistent recovery or rollback. GT.M processes constantly cooperate with each other to write the data from the buffers to the secondary storage. At epoch time, GT.M holds a critical section to complete all pending write operations. Other processes updating the database wait till GT.M completes the epoch. GT.M epoch events have a slightly higher(possibly spiky) impact. With -EPOCHTAPER, GT.M tries to minimize the epoch duration by reducing the amount of data to flush as an epoch event approaches.

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.

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.

Prior Generation Journal Files
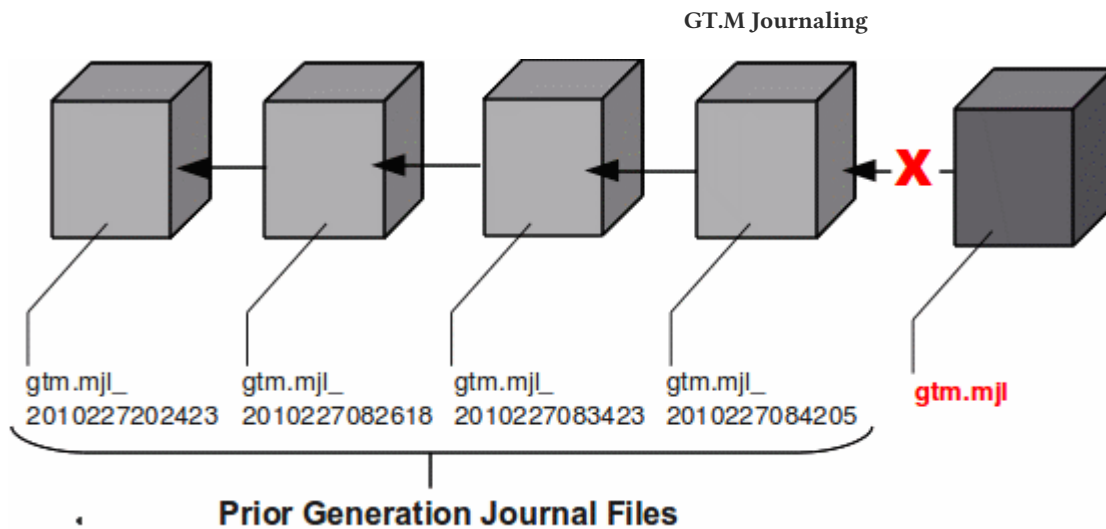
Because -NEWJNLFILES=NOPREVLINK cuts back link of the newly created journal file, any subsequent recovery, rollback or request to the source server go back past this discontinuity.

> **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...)] TSTART can manage local variable state on restarts.	**ZTS[TART][:tvexpr]**
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 conjuction 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 197)) 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 under unusual circumstances 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 88).

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 database files, journal files, regions or replication state.

The format for the MUPIP SET command is:

```
MUPIP SE[T] -qualifier... {-F[ILE] file-name|-JN[LFILE journal-file|-REG[ION] region-list}
```

The file-specification, journal 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]
-JN[LFILE]
-R[EGION]
```

-FILE and -REGION act together with one or more of the SET action qualifiers:

```
-[NO]JOURNAL[=journal-option-list] -REPLICATION=<replication-option>'
```

SETObject Identifying Qualifiers

The following qualifiers identify the journaling targets:

```
-F[ILE]
```

Specifies that the argument to the SET is a file-specification for a single database file. A Journal file's name can include UTF-8 characters.

Old journal files stay open for about 10 seconds after a switch to a new journal file.

```
-R[EGION]
```

Specifies 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 and -JNLFILE qualifiers.

```
-JN[LFILE]
```

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.

-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.

-dbfilename=*file_name*

Associates a journal file with a different database file; this command may be useful in arranging unusual RECOVER or ROLLBACK scenarios.

-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.

-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.

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 88).

-[NO]J[OURNAL][=*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.

- -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 197).

 -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.

- 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 at multiples of ALIGNSIZE (except at offset 0 which houses the journal file header) always have a valid journal record. In order to ensure this, the GT.M run-time system, as needed, writes padding data 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 from a corrupt journal file as possible.

- While a larger align size trades 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 infrequent 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 using the same journal file(s).

ALL[OCATION]=*blocks*

ALLOCATION is the size at which GT.M should start checking for free space.

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.

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 and 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 informational message. GT.M produces an error when the rounded value of AUTOSWITCHLIMIT is less that 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 JNLALLOCGROW 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.

> **Important**
>
> AUTOSWITCHLIMIT is always the sum of ALLOCATION and a multiple of EXTENSION value.
> GT.M automatically attempts to adjust ALLOCATION/AUTOSWITCHLIMIT values whenever
> AUTOSWITCHLIMIT is not the sum of ALLOCATION and a multiple of the EXTENSION value. When a
> MUPIP SET -JOURNAL command results in an inappropriate automatic adjustment (upward or downward),
> GT.M rejects the command and reports a non-zero exit status.

[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 2312 blocks. The minimum BUFFER_SIZE is 2307 blocks. The maximum BUFFER_SIZE is 1Mi blocks which
means that the maximum buffer you can set for your journal file output is 512MiB.

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.

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*

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.

When a journal file reaches the size of ALLOCATION and any multiple of EXTENSION, GT.M checks for free space on the file system.

- If the available space is less than three times the EXTENSION, GT.M sends the DSKSPACEFLOW informational message to the operator log.

- If the available space is less than EXTENSION (even if there is space to continue journaling), GT.M shuts off journaling for the corresponding database file or institutes and Instance Freeze.

In addition to checking for free space on the file system, GT.M also attempts to write the JNLSPACELOW message to the operator log three times as a journal file reaches its maximum size. The first JNLSPACELOW message appears in the operator log when twice the number of EXTENSION blocks are left before a journal file reaches the maximum size (AUTOSWITCHLIMIT), the second appears when EXTENSION blocks are left, and the third appears when the journal file reaches the maximum size (AUTOSWITCHLIMIT).

> **Important**
>
> While JNLSPACELOW messages should be used as an operational aid for monitoring the journal file growth and planning for providing sufficient disk space for the next generation journal files, the DSKSPACEFLOW message indicates that the available free disk space is low on the file system and requires immediate operator intervention to provide enough disk space to allow GT.M to continue journaling. If GT.M cannot continue journaling, it turns journaling off or freezes the system, depending on whether your configuration has Instance Freeze enabled. Use a combination of ALLOCATION, AUTOSWITCHLIMIT, and EXTENSION values to setup a threshold point for triggering the JNLSPACELOW messages to the operator log. The operator log monitoring mechanism for the JNLSPACELOW messages should include actions to ensure that adequate disk space is available for the next generations of the journal file.

With EXTENSION=0, GT.M checks for free space on the file system two times - The first check happens when the journal file reaches the ALLOCATION size and second happens when the journal file reaches the maximum size (AUTOSWITCHLIMIT). If EXTENSION=AUTOSWITCHLIMIT, the check for free space happens only once. With EXTENSION=0, GT.M writes the JNLSPACELOW message to the operator log only once when the journal file reaches its maximum size. This bypasses the operational aid that JNLSPACELOW messages may provide to help ensure that adequate disk space is available for the next generations of the journal file.

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 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.

- 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 UTF-8 characters.

> **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.

[NO]S[YNC_IO]

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" [230] 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
```

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 AGREG 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 76).

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	Only one	One or more	Only one	One or more	One or more
-EX[TRACT] [=file specification]	-BA[CKWARD] -FO[RWARD]	-A[FTER]=time	-FET[CHRESYNC]= port number	-[NO]	-G[LOBAL]=global list
-REC[OVER]		-BE[FORE]=time	-RES[YNC]=jnl-sequence number	AP[PLY_AFTER_IMAGES]	-ID=pid-list
-RO[LLBACK]		-[NO]LOO[KBACK_LIMIT][=lookback option list]		-[NO]BR[OKENTRANS]][=file name]	-T[RANSACTION]= transaction type
-SH[OW][=show option list]		-SI[NCE]=time		-[NO]CHA[IN]	-U[SER]=user-list
-[NO]V[ERIFY]				-[NO]CHE[CKTN]	
				-[NO]ER[ROR_LIMIT] [=integer]	
				-FE[NCES]=fence option	
				-FU[LL]	
				-[NO]IN[TERACTIVE]	
				-[NO]LOST[TRANS] [=file name]	
				-[NO]O[NLINE]	
				-RED[IRECT]=file pair list	
				-VERB[OSE]	
				-DE[TAIL]	

Also ensure that you adhere to the following rules:

1. -AFTER is incompatible with -RECOVER or -ROLLBACK; that is -AFTER requires -FORWARD, and only applies to action qualifiers: -EXTRACT, -SHOW, and -VERIFY.

2. -APPLY_AFTER_IMAGE is compatible only with -RECOVER, or -ROLLBACK.

3. -BACKWARD is incompatible with -FORWARD, -AFTER, -CHECKTN, -NOCHAIN, and -REDIRECT.

4. -[NO]BROKENTRANS is compatible only with -RECOVER, -ROLLBACK, or -EXTRACT.

5. -CHAIN is only compatible with -FORWARD.

6. -CHECKTN is incompatible with -BACKWARD.

7. -DETAIL is compatible only with -EXTRACT.

8. -FETCHRESYNC is only compatible with the -ROLLBACK action in the -FORWARD direction and is incompatible with RESYNC.

9. -FORWARD is incompatible with -BACKWARD, -FETCHRESYNC, -LOOKBACK_LIMIT, -ONLINE and -SINCE.

10. -FULL is compatible only with -EXTRACT, -SHOW, or -VERIFY.

11. -[NO]LOSTTRANS is compatible only with -RECOVER, -ROLLBACK, or -EXTRACT.

12. -REDIRECT is compatible only with -BACKWARD and -RECOVER.

13. -RESYNC is only compatible with the -ROLLBACK action and incompatible with FETCHRESYNC.

14. -ROLLBACK is incompatible with -RECOVER, -CHAIN, -CHECKTN, -REDIRECT, time qualifiers of -SHOW except -BEFORE.

15. -SINCE is incompatible with -FORWARD.

16. -TRANSACTION is compatible only with -EXTRACT and -SHOW.

17. -USER is compatible only with -EXTRACT and -SHOW.

18. file list must not be asterisk (*) for -REDIRECT.

19. file list must be asterisk (*) for -BACKWARD -ROLLBACK; -ROLLBACK -FORWARD accepts a list of journal file names.

20. Journal selection qualifiers are incompatible with -RECOVER, -ROLLBACK, and -VERIFY.

21. If -BEFORE (time-based) and -FETCHRESYNC/-RESYNC (sequence-number-based) are specified in the same MUPIP JOURNAL -ROLLBACK command, the qualifier that corresponds to an earlier database state or point in time prevails. For example, -BEFORE prevails when the update corresponding to the sequence number obtained through the -FETCHRESYNC command happened at a later time relative to the -BEFORE qualifier and vice versa.

22. -FETCHRESYNC, -ONLINE, and -RSYNC_STRM qualifiers are not compatible with -ROLLBACK -FORWARD.

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 requires an inactive journal file that is available for exclusive (standalone) use.

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> may also be the name of a currently open named pipe (FIFO) set up to receive the extracted records.

<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 218) 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 218) 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.

-PARA[LLEL][=n]

PARA[LLEL][=n] specifies the number of parallel threads (for backward processing) and parallel processes (for forward processing). Parallel threads typically increase the speed of MUPIP JOURNAL RECOVER/ROLLBACK operations.

Omitting the qualifier or specifying a value of one (1) defaults to a single process with no threads. Omitting the value or specifying a value of zero (0) specifies one thread or process per region.

A value greater than one (1) specifies the maximum number of concurrent threads or processes MUPIP should use, although it never uses more than one per region. If the number of regions exceeds the specified value, MUPIP allocates one thread or processes in an order determined by timestamps in the journal records.

The environment variable gtm_mupjnl_parallel provides a value when the MUPIP JOURNAL command has no explicit -PARALLEL qualifier; when defined with no value gtm_mupjnl_parallel acts like -PARALLEL with no value. When the -PARALLEL qualifier (or the gtm_mupjnl_parallel environment variable) specifies the use of parallel processes in the forward phase of a MUPIP JOURNAL command, MUPIP may create temporary shared memory segments and/or extract files (corresponding to -extract or -losttrans or -brokentrans qualifiers) and clean these up at the end of the command; however an abnormal termination such as a kill -9 might cause these to be orphaned. Journal extract files (created by specifying one of -extract or -brokentrans or -losttrans to a MUPIP JOURNAL command) contain journal records sorted in the exact order their corresponding updates happened in time.

-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 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 218). MUPIP JOURNAL does not allow any two of -EXTRACT, -LOSTTRANS or -BROKENTRANS to specify the same file name unless they are special files (-stdout or /dev/null).

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 datatabase 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 neccessary, 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_back* files and process them using a M program.

> ### Note
>
> Using -RECOVER on a replicated database initiates database recovery but turns replication OFF. Under most circumstances, there is no need to perform a -RECOVER operation on replicated regions.

-ROLLBACK [{-ON[LINE]|-NOO[NLINE]}]

-ROLLBACK -FORWARD "*" command does what a -RECOVER -FORWARD "*" would do except that the ROLLBACK also updates sequence number related fields in the database file header and ensures update serialization across regions. -RECOVER can leave one database region ahead of another region. -RECOVER cannot ensure database Consistency across regions whereas -ROLLBACK can.

When used without time qualifiers, -ROLLBACK -FORWARD "*" applies update records in journal files to backed up copies of database files to bring them to the same state that -ROLLBACK -BACKWARD "*" would bring crashed database files. Note that, in the context of -RECOVER and -ROLLBACK, the "*" indicates the use of all the appropriate journal files in all the replicated regions and the quotes prevent inappropriate expansion by the OS shell.

Databases recovered with -ROLLBACK can be used in replicated instances.

> ⚠️
> - -ROLLBACK -FORWARD leaves the journaling state turned off in database files (as does MUPIP JOURNAL -RECOVER -FORWARD), which in turn means that replication is also turned off; re-enable journaling, and turn replication on, before using database files in environments where they can be updated, but journaling and replication may be left off if subsequent access is read-only.
>
> - After a -ROLLBACK -FORWARD, recreate the replication instance file as part of turning replication on for the recovered database.
>
> - -ROLLBACK -FORWARD can use both before-image and nobefore-image journal files.

-ROLLBACK initiates the central JOURNAL operation for a replicated database. MUPIP JOURNAL commands may specify -ROLLBACK with other action qualifiers but not with -RECOVER. With -BACKWARD, if you do not specify -BEFORE or -FETCHRESYNC, the database rolls back to the last consistent state. With -BACKWARD, the command allows only an asterisk (*) argument for the journal file selection, that is, -ROLLBACK selects journal files by itself.

If a transaction is found with incomplete fence, it is considered incomplete or broken.

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).

-ROLLBACK -BACKWARD exits with an error message if a database does not have both journaling and replication either enabled or disabled.

> **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.

-NOO[NLINE]

Specifies that ROLLBACK requires exclusive access to the database and the replication instance file, which means the database and the replication instance files are inaccessible during a -ROLLBACK -NOONLINE.

-ROLLBACK -FORWARD does not support the -[NO]O[NLINE] qualifier.

-ON[LINE]

Specifies that ROLLBACK can run without requiring exclusive access to the database and the replication instance file.

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 19).

> **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.

By default, MUPIP JOURNAL -ROLLBACK -BACKWARD is -NOONLINE.

GT.M increments ISV $ZONLNRLBK every time a process detects a concurrent MUPIP JOURNAL -ONLINE -ROLLBACK.

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, the ROLLBACK only removes any incompletely committed TP transactions or non-TP mini-transactions; any concurrent 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:

- For the duration of the rollback, replication is turned OFF on all regions and turned back ON at the end of the rollback.

- -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.

-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 MUPIP 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/V6.3-002_x86/g/gtm.mjl
-----------------------------------------------------------------------------
Journal file name       /home/jdoe/.fis-gtm/V6.3-002_x86/g/gtm.mjl
Journal file label      GDSJNL23
Database file name      /home/jdoe/.fis-gtm/V6.3-002_x86/g/gtm.dat
 Prev journal file name /home/jdoe/.fis-gtm/V6.3-002_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
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
-----------------------------------------------------------
0000006706 jdoe-laptop jdoe          0    2012/11/06 17:30:33
Process That First Opened the Journal File:
PID         NODE         USER        TERM JPV_TIME
-----------------------------------------------------------
0000006706 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/V6.3-002_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 journal files for integrity. This qualifier cannot have a value. -VERIFY scans journal files and checks if they have legal form, if not, it terminates without affecting the database files.

-NOVERIFY is the default for -RECOVER -FORWARD and -ROLLBACK -FORWARD. -VERIFY is the default for RECOVER -FORWARD -NOCHECKTN. -VERIFY is also the default for all other MUPIP JOURNAL commands (including -RECOVER - BACKWARD and -ROLLBACK -BACKWARD).

-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 (the default) when -BACKWARD is used with -RECOVER or -ROLLBACK. For -FORWARD, unless there is reason to suspect that the journal files have sustained structural damage, FIS suggests the use of -NOVERIFY (the default).

When used independent of -RECOVER (or -ROLLBACK), -[NO]VERIFY option does not need database access. In this case 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.

`-FO[RWARD]`

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 208) 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 except -BEFORE are incompatible with -ROLLBACK.

The following section describes the time qualifiers in more detail:

```
-A[FTER]=time
```

Specifies the starting time stamp in the journal file after which any -FORWARD action should start processing. This time qualifier is compatible only with -EXTRACT,-SHOW, or -VERIFY.

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 MUPIP displays a warning message. 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 the ending time for any action -FORWARD or -BACKWARD. The ending time is inclusive, that is, -BEFORE includes the records that match the specified ending time in the journal records. 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 MUPIP displays a warning message.

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.

When used with -ROLLBACK or -RECOVER, -BEFORE specifies the time at which MUPIP stops applying updates to the database in its forward processing phase (i.e., no journal records with update times after the -BEFORE time are applied to the database).

When both -FETCHRESYNC/-RESYNC and -BEFORE are used with -ROLLBACK -BACKWARD, the qualifier corresponding to an earlier database state or point in time prevails. For example, -BEFORE prevails when the update corresponding to the sequence number obtained through the -FETCHRESYNC command happened at a later time relative -BEFORE and vice versa.

```
-[NO]LOO[KBACK_LIMIT][=lookback-option-list]
```

Specifies how far JOURNAL -RECOVER -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
```

The -SINCE time qualifier applies to MUPIP JOURNAL -BACKWARD. The -SINCE qualifier specifies how far back in time MUPIP JOURNAL should at least process (from the end of the journal file), before starting the forward processing. The actual turn-around point for -RECOVER and -ROLLBACK in each database region is an epoch in the journal files before or at the -SINCE time, but not after it.

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

The following qualifier is compatible only with -EXTRACT.

```
-S[EQNO]=<sequence_number_list>
```

Specifies a list of sequence numbers to include or exclude in the journal extract. <sequence_number_list> is a comma separated list of sequence number(s) in decimal form. When a sequence number has a (~) prefix, -SEQNO excludes it from the journal extract. For replicated regions, EXTRACT -SEQNO uses replication sequence numbers, which may select records from multiple regions. For unreplicated regions, EXTRACT uses journal sequence numbers, but specifying sequence number selection with more than one regions produces a JNLEXTRCTSEQNO error. When the sequence number list contains a sequence number involved in a TP transaction, EXTRACT reports it in a broken transaction file when the result does not contain all regions, which is commonly the case without replication, and may be the case with replication when not all regions are available to the utility.

Example:

$mupip journal -extract -seqno="~1,2,3,4,~5" -forward -broken=trans.broken -lost=trans.lost "*"

This example produces a journal extract containing journal sequence numbers 2,3,and 4. 1 and 5 are not part of the journal extract as they have the (~) prefix.

The following qualifiers are compatible only with -ROLLBACK.

`-FET[CHRESYNC]`

For more information on -FETCHRESYNC, refer to "Rolling Back a Replicated Database " (page 321).

`-RES[YNC]=<journal sequence number>`

For more information on -RESYNC, refer to "Rolling Back a Replicated Database " (page 321).

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 -RECOVERY or -ROLLBACK. AIMG are "snapshots" of the database updates captured by GT.M 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.

`-[NO]BR[OKENTRANS]=<extract file>`

-[NO]BROKENTRANS is an optional qualifier for -ROLLBACK, -RECOVER and -EXTRACT. NOBROKENTRANS suppresses the generation of a broken transaction file. Otherwise, if the command does not specify a file name and MUPIP finds any broken transactions, MUPIP JOURNAL creates a broken transaction file 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 begining 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 ending transaction number of every journal file is equal to the begining 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. MUPIP performs -VERIFY when -NOCHECKTN is specified. Use -NOCHECKTN with caution because it may lead to integrity issues in the recovered database and journal files.

ROLLBACK -FORWARD accepts only -CHECKTN, which is the default, but does not accept -NOCHECKTN.

-CHECKTN is incompatible with -BACKWARD.

```
-[CO]RRUPTDB
```

Extracts journal records into a single file even if the database is corrupt or missing. Always specify a journal file name when you are using -CORRUPTDB. -CORRUPTDB does not recognize the wildcard character "*" for journal file name and is incompatible with -FENCES, -LOST, and -BROKEN qualifiers.

```
-[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 journal processing finds a broken transaction, 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 command followed by a TCOMMIT command. All updates between a TSTART and a TCOMMIT 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/-ROLLBACK are not case-sensitive.

The fence options are:

- **NONE**

  This causes MUPIP JOURNAL -RECOVER to apply all individual updates as if transaction fences did not exist. Note that, this means journal processing treats a SET/KILL within a TP transaction as if it was an unfenced SET/KILL. -FENCES=NONE is not permitted for MUPIP JOURNAL -ROLLBACK.

- **ALWAYS**

  This causes MUPIP JOURNAL -RECOVER to treat any unfenced or improperly fenced updates as broken transactions. FENCES=ALWAYS is not permitted for MUPIP JOURNAL -ROLLBACK.

- **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. 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. -ROLLBACK accepts -FENCES=PROCESS, which is the default.

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 208) or "-ROLLBACK [{-ON[LINE]|-NOO[NLINE]}] " (page 210) 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, MUPIP extracts the entire journal file contents (including those records that were rolled back). 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 applies when the MUPIP command is entered from a terminal. The default is -INTERACTIVE.

`-[NO]LOST[TRANS]=<extract file>`

-[NO]LOSTTRANS is an optional qualifier for -RECOVER, -ROLLBACK and -EXTRACT. NOLOSTTANS suppresses the generation of a broken transaction file. Otherwise, if the command does not specify a file name and MUPIP finds any lost transactions, MUPIP JOURNAL creates a lost transaction file using the name of the current journal file being processed with a .lost extension.

Journal processing treats any complete transactions after a broken transaction as a lost transaction, and writes such transactions into the lost transaction file. -RECOVER might consider it as good transaction and apply it to the database, if -ERROR_LIMIT qualifier allows it to do so.

Note that, if selection qualifiers are specified, journal processing does the broken transaction determination (and therefore lost transaction determination as well) 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 results in every journal record being considered complete and hence preventing 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="bgdbb.dat=test.dat" bgdbb.mjl
```

This JOURNAL command does a forward recovery that -REDIRECTs the updates in bgdbb.mjl from bgdbb.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 (,) seperated list of values.

```
-G[LOBAL]=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 tilda 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 tilda 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.

```
-GV[PATFILE]=/path/to/pattern-file
```

Specifies the location of a pattern file containing a list of patterns for all types of SET journal records that MUPIP JOURNAL -EXTRACT should include or exclude from processing. Use this qualifier to restrict the output of a journal extract by global nodes value (that is, by all types of SET records).

The following details the syntax of the pattern file, and examples of how MUPIP JOURNAL -EXTRACT responds:

- When a pattern entry starts with a tilda sign (~), -GVPATFILE excludes the matching global node values from the JOURNAL EXTRACT file. For example, ~(not this value) excludes all globals that exactly match " not this value".

- When the pattern does not start a tilda sign (~) or contain an asterisk (*), MUPIP JOURNAL -EXTRACT reports only those global node values that exactly match the pattern. For example: " match this value".

- When a pattern contains an asterisk (*), MUPIP JOURNAL -EXTRACT expands it and tries to match multiple characters.

  Example : "*a*b*" matches values like "ab", "..ab", "ab.. ", "a..b", "aaabbabb", and so on but does not match values like "ba", "aaa", "bbb", and so on.

- When a pattern contains a percentage (%), MUPIP JOURNAL -EXTRACT matches it for one character. Example : "a%b%" matches values like "a1b1" but does not match values like "ab", "aabbc", and so on.

- A pattern can be enclosed within parentheses "()" for readability.

- When you use any of the following characters in the pattern, you can escape them by preceding the character with "\". Example : "a**b" matches values like "a*..b" but not "a..b".:

  - "(" and "~" at the beginning.

  - ")" at the end.

  - "\", "*", and "%" occurring anywhere within the pattern.

- In UTF-8 mode, the contents of the pattern file can include Unicode characters.

- If a pattern file does not exist, MUPIP JOURNAL -EXTRACT produces the FILEOPENFAIL error and returns a non-zero exit code to the shell.

You can specify multi-line entries in a pattern file.With multiple lines, MUPIP JOURNAL EXTRACT produces those SET records that match any one of the pattern lines with the exception of exclusion patterns (those starting ~) which take precedence over other non-exclusion patterns.

Here are a few examples of the pattern file, and how MUPIP JOURNAL -EXTRACT matches the pattern file values:

```
> cat matchA_notAA.txt
~(*AA*)
*A*
> $gtm_dist/mupip journal -extract -gvpatfile=matchA_notAA.txt -forward "*"
Extracts global values that contain a single "A", but not "AA".
> cat ending22.txt
*notmatching*
*22
> $gtm_dist/mupip journal -extract -gvpatfile=ending22.txt -forward "*"
Extracts global values ending with "22", even when there are no globals containing "notmatching".
> cat startswithsplchars.txt
\**
\~*
> $gtm_dist/mupip journal -extract -gvpatfile=matchA_notAA.txt -forward "*"
Extracts global values that start with a "*" or a "~".
```

```
-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[RANSACTION]=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 exmaple, -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.

JournalExtract Formats

Journal EXTRACT files always start with a label to identify the format of the file. Simple and detail EXTRACT files have different labels, and the labels are specific to a GT.M release.

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.

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 230)). When replication is not in use and the transaction is a TP transaction, token_seq is an 8-byte token that uniquely identifies the entire TP transaction. For non-replicated, non-TP 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
LGTRIG
▶ 13\time\tnum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\trigdefinition
```

▼

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.

13 record indicates a trigger definition as a logical action from an originating/primary instance to a replicating/secondary instance

The listing below shows an example of a simple journal extract:

```
GDSJEX07
01\65457,9380\1099511627777\3664\zappa\scheese\\0\\\
02\65457,9380\1099511627777\3664\0
03\65457,9380\1099511627777\3664\0\1
```

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, 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\nnam\unam\term\clntpid\clntnnam\clntunam\clntterm
PINI(V)
▶ time\tnum\chksum\pid\nnam\unam\term\mode\logintime\image_count\pname\clntpid\clntnnam\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
TLGTRIG
   time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\trigdefinition
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
ULGTRIG
   time\tnum\chksum\pid\clntpid\token_seq\strm_num\strm_seq\updnum\trigdefinition
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\chksum\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.

GT.M Journaling

Legend (All hexadecimal fields have a 0x prefix. All numeric fields otherwise are decimal):

tnum	Transaction number
chksum	Checksum for the record.
fully_upgraded	1 if the db was fully upgraded (indicated by a dse dump -file -all) at the time of writing the EPOCH
pid	Process id that wrote the jnl record.
clntpid	If non-zero, clntpid is the process id of the GT.CM client that initiated this update on the server side.
jsnum	Journal sequence number.
token	Unique 8-byte token.
strm_num	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.
strm_seq	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.
tid	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 **tid** to null. TRANSACTIONID can specify any value for **tid** but affects GT.M behavior only when TRANSACTIONID specifies BATCH or BA.
token_seq	If replication is turned on, it is the journal sequence number. If not, it is a unique 8-byte token.
trigdefinition	Trigger definition string corresponding to an LGTRIG journal record.
updnum	=n where this is the nth update in the TP transaction. n=1 for the 1st update etc. 0 for non-TP.
nodeflags	Decimal number interpreted as a binary mask.. Currently only 5 bits are used. • 00001 (1) => update journaled but NOT replicated (For example, update inside a trigger) • 00010 (2) => update to a global that had at least one trigger defined, even if no trigger matched this update • 00100 (4) => $ZTWORMHOLE holds the empty string ("") at the time of this update or was not referenced during this update • 01000 (8) => update did not invoke any triggers even if they existed (For example, MUPIP LOAD) • 10000 (16) => 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. Combinations of the above bits would mean each of the individual bit characteristics. For example, 00011 => update within a trigger context, 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.

node	Key that is being updated in a SET or KILL.
sarg	Right-hand side argument to the SET (that is, the value that the key is being SET to).
partners	Number of journaled regions participating in this TP (TCOM/ZTCOM record written in this TP) .
opcode	Inctn opcode. See gdsfhead.h inctn_opcode_t for all possible values.
blknum	Block number corresponding to a PBLK or AIMG or INCTN record.
bsiz	Block size from the header field of a PBLK or AIMG record.
blkhdrtn	Transaction number from the block header of a PBLK or AIMG record.
ondskbver	On disk block version of this block at the time of writing the PBLK or AIMG record. 0 => V4, 1 => V5.
incdetail	0 if opcode=1,2,3; blks2upgrd if opcode=4,5,6; blknum if opcode=7,8,9,10,11,12,13
ztwormhole	string corresponding to $ZTWORMHOLE
blks2upgrd	# of new V4 format bitmap blocks created if opcode=4,5; csd->blks_to_upgrd if opcode=6
uname	Name of the user that wrote this PINI record.
clntunam	If non-empty, clntunam is the name of the GT.CM client that initiated this update on the server side.

After the header(s), each line of a detail journal extract file begins with two hexadecimal numbers. The first of these is the offset in the journal file of the record being printed, and the second (enclosed in square brackets) is the length of that record. The listing below shows an example of a detail extract:

```
GDSJDX08
0x00010000 [0x00d0] :: PINI
 ▶ \65457,9380\1099511627777\2480956899\3664\zappa\scheese\\0\\\
0x000100d0 [0x00b8] :: EPOCH
 ▶ \65457,9380\1099511627777\973009291\3664\0\1\0\96\101\1
0x00010188 [0x0020] :: PFIN   \65457,9380\1099511627777\3421761717\3664\0
0x000101a8 [0x0028] :: EOF
 ▶ \65457,9380\1099511627777\884316091\3664\0\1
```

▼

Chapter 7. Database Replication

Revision History		
Revision V7.0-003	24 June 2022	• In "Starting the Source Server" (page 297), remove information about compiling zlib. Warn about TLSv1.3 and renegotiation vs key update
Revision V7.0-002	29 March 2022	• In "Starting the Source Server" (page 297), correct formatting
Revision V7.0-001	24 November 2021	• In "Starting the Source Server" (page 297), improve the description of -RENEGOTIATE_INTERVAL. • In "Starting the Receiver Server " (page 314), Connection restarts with -updateresync • In "Starting the Update Process" (page 318), Starting replication with no helper processes
Revision V7.0-000	12 February 2021	• In "Reporting the Current Backlog of Journal Records" (page 310), add information about the acknowledgement sequence number and examples • In "Shutting down the Source Server" (page 305), add information about -ZEROBACKLOG and -TIMEOUT.
Revision V6.3-014	06 October 2020	• In "Creating the Replication Instance File" (page 294), add reference to gtm_repl_instname and gtm_repl_instance. • In "Switchover possibilities in an A→B replication configuration" (page 268), fixed default wait period to 30 seconds • In "Shutting down the Source Server" (page 305), added the default and max value of -timeout and information about the internal max wait time. • In "Starting the Source Server" (page 297), add -CONNECTPARAMS information and adjust the maximum -BUFFSIZE
Revision V6.3-013	30 June 2020	• In "Setting up an A→B→C replication configuration with empty databases" (page 265), Minor formatting adjustment • In "Setting up an A→B replication configuration with empty databases" (page 264), Minor formatting adjustment • In "Setting up an A→P replication configuration with empty databases" (page 266), formatting adjustment of the note.

		• In "Starting the Source Server" (page 297), Adjust replication -buffsize maximum
Revision V6.3-011	20 December 2019	• In "Instance Bound Global Directories " (page 240), simplify the content to encourage the use of replication and instance bound global directories.
		• In "Introduction" (page 236), content improvement
Revision V6.3-009	27 June 2019	• In "Replication Architecture " (page 247), add more information about the purpose of a passive Source Server and general content improvements.
		• In "Freezing an Instance" (page 308), add new section.
		• In "Replication Instance File" (page 250), add a reference to autorollback
		• In "Starting the Source Server" (page 297), specify the default value for -connectparams; content improvements; organize qualifiers by relevance.
		• In "Stopping the Receiver Filter" (page 307), add new section.
		• In "Stopping the Source Filter" (page 307), add -instsecondary qualifier to specify the Source Server target for stopping the Source filter.
		• In "Starting the Receiver Server " (page 314), organize qualifiers by relevance.
Revision V6.3-008	24 April 2019	• In "Checking Server Health" (page 308), source -checkhealth does not accept -helpers option
		• In "Checking Server Health" (page 320), receiver -checkhealth supports -helpers option
		• In "Download Replication Examples" (page 259), remove redundant material.
		• In "Starting the Source Server" (page 297), specify that MUPIP ignores the -BUFFSIZE qualifier when the Journal Pool is already set up.
		• In "TLS/SSL Replication" (page 256), remove redundant material.
		• In "Setting up a secured TLS replication connection" (page 286), remove redundant material.
Revision V6.3-007	04 February 2019	• In "Replicating Instance Starts from Backup of Originating Instance (A→B and A→P)" (page 267), Use PREVLINK when taking a backup to be used exclusively for replication.

		• In "Rolling Back a Replicated Database " (page 321), correct the description of -FETCHRESYNC. • In "Starting the Receiver Server " (page 314), improve the description of -AUTOROLLBACK.
Revision V6.3-006	26 October 2018	• In "Displaying/Changing the attributes of Replication Instance File and Journal Pool" (page 295), add information about the -cleanslot qualifier. • In "Introduction" (page 236), correct the alignment of bc_repl.svg. • In "Setting up a secured TLS replication connection" (page 286), make corrections and add clarification about openssl.conf
Revision V6.3-005	03 July 2018	• In "Why do we need a $gtmcrypt_config file?" (page 514), Disable SSLv3 • In "Download Replication Examples" (page 259), adjust formatting to fix a PDF rendering issue • In "Instance Bound Global Directories " (page 240), fix a typo.
Revision V6.3-004	23 March 2018	• In "Rollback data from crashed (idle) regions" (page 289), add procedure to rollback data from crashed (idle) regions • In "Starting the Source Server" (page 297), specify that the Source Server directs errors to the log file, add fixes for the filter example
Revision V6.3-003	12 December 2017	• In "Comparison other than Recovery" (page 258), correct incomplete statement • In "Download Replication Examples" (page 259), changing script title to env and spell change • In "Instance Bound Global Directories " (page 240), add new section • In "Replication Instance File" (page 250), instance reconnect on the basis of most recent shared journal seq number • In "Recovery" (page 257), corrections and improvements • In "Starting the Source Server" (page 297), correct the quoting of the -filter qualifier and remove -replicate • In "Introduction" (page 236), add information about using extended global references or SET $ZGBLDIR to update another source instance.

		• In "Creating a new Replication Instance File" (page 285), Updated GTM version
Revision V6.3-002	22 August 2017	• In "Instance Freeze" (page 254), add that MUPIP REPLICATE -SOURCE -JNLPOOL -SHOW displays the load status of the custom errors file.
Revision V6.3-001	20 March 2017	• In "A and P require rollback" (page 271), used capitals for keywords • In "Why do we need a $gtmcrypt_config file?" (page 514), fixed a minor typo • In "Creating the Replication Instance File" (page 294), added the description of the -noqdbrundown qualifier. • In "Displaying/Changing the attributes of Replication Instance File and Journal Pool" (page 295), added the description of the -noqdbrundown qualifier. • In "Changing the global directory in an A→B replication configuration" (page 278), added a new procedures for changing global directory settings in an A->B replication configuration; minor tweak to wording • In "Helper Processes" (page 249), specified that Helper Processes increase replication throughput, decrease backlog, and improve manageability. • In "Instance Freeze" (page 254), removed the reference to the Instance Freeze as field test grade implementation. The Instance Freeze facility became production grade in V6.0-001. • In "Procedures" (page 259), rewrote most sections and added downloadable replication example scripts. • In "Switchover possibilities in a B←A→P→Q replication configuration " (page 276), updated for V6.3-000A • In "Replication Instance File" (page 250), used caps for keywords • In "Recovery" (page 257), used capitals for keywords and leading delimiters for qualifiers • In "Replicating Instance Starts from Backup of Originating Instance (A→B and A→P)" (page 267), followed convention of capitalizing key words when they are outside of examples • In "Limitations - SI Replication" (page 246), removed reference to OpenVMS

		• In "Shutting down the Source Server" (page 305), added the description of the -zerobacklog qualifier.
		• In "Examples" (page 241), used capitals for keywords outside of examples
		• In "Starting the Source Server" (page 297), removed references to Solaris, HP-UX, and OpenVMS; specified that pool limit is 4GiB - 8; specified that GT.M supports the use of IBM provided zlib library for AIX.
		• In "Stopping the the Receiver Server and/or the Update Process" (page 319), added information about the -shutdown qualifier; alphabetized the qualifiers after -shutdown.
		• In "Introduction" (page 236), language cleanup - intermediate save
		• In "Setting up a secured TLS replication connection" (page 286), Changed Helen and Phil to Alice and Bob.
		• In "Creating a new Replication Instance File" (page 285), changed 4.5-000 to 5.4-000
		• In "Setting up a new replicating instance of an originating instance (A→B, P→Q, or A→P)" (page 290), changed on -> one\n
Revision V6.2-002		Added examples and improved the instructions in "Procedures" (page 259).
Revision V6.2-001	27 February 2015	In "Starting the Source Server" (page 297), added the NOJNLFILEONLY qualifier.
Revision V6.1-000/1	04 September 2014	In "Procedures" (page 259), corrected the downloadable scripts example (msr_proc2.tar.gz) for setting up an A→P replication configuration.
Revision V6.1-000	01 August 2014	• In "Starting the Source Server" (page 297), added the description of -TLSID, -[NO]PLAINTEXTFALLBACK, and -RENEGOTIATE_INTERVAL qualifiers.
		• In "Starting the Receiver Server " (page 314), added the descriptions of -TLSID, -START, and -LOG qualifiers..
		• In "Procedures" (page 259), added a new topic called "Setting up a secured TLS replication connection" (page 286). This topic includes a downloadable example that creates two databases and sets up TLS replication between them.
		• Added a new section called "TLS/SSL Replication" (page 256).
Revision V6.0-003/1	19 February 2014	• In "Starting the Source Server" (page 297), added information about the use of the gtm_ipv4_only environment variable.

		• In "REORG " (page 136), added a caution note about running successive reorgs.
Revision V6.0-003	27 January 2014	• In "Procedures" (page 259), added downloadable script examples for A→B and A→P replication scenarios. • In "Activating a Passive Source Server" (page 305), added information about the ACTIVE_REQUESTED mode. • In "Deactivating an Active Source Server" (page 306), added information about the PASSIVE_REQUESTED mode. • In "Starting the Source Server" (page 297), added information about the IPv6 support.
Revision V6.0-001/1	22 March 1013	Improved the formatting of all command syntaxes and corrected the description of the -helper qualifier.

Revision V6.0-001	27 February 2013	• In "Instance Freeze" [254], added a note about the behavior when a process configured for instance freeze encounters an error with journaling.
		
		• Improved the description of -UPDATERESYNC.
		
		• In "Procedures" [259], added the following topics:
		
		  • "Setting up a new replicating instance of an originating instance (A→B, P→Q, or A→P)" [290]
		
		  • "Replacing the replication instance file of a replicating instance (A→B and P→Q)" [290]
		
		  • "Replacing the replication instance file of a replicating instance (A→P)" [291]
		
		  • "Setting up a new replicating instance from a backup of the originating instance (A→P)" [291]
		
		• In "Displaying/Changing the attributes of Replication Instance File and Journal Pool" [295], corrected the description of the -offset and -size qualifiers of -EDITINSTANCE.

Revision V6.0-000/1	21 November 2012	• In "Commands and Qualifiers" [292], added the description of the -log_interval qualifier. • In "Starting the Receiver Server " [314], added the description of the -initialize qualifier and updated the description of -updateresync for V6.0-000.
Revision V6.0-000	19 October 2012	• In "Starting the Source Server" [297], added the description of the -freeze qualifier.

		• Added a new section called "Instance Freeze" (page 254).

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 helps provide business continuity for GT.M databases. 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 any 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.
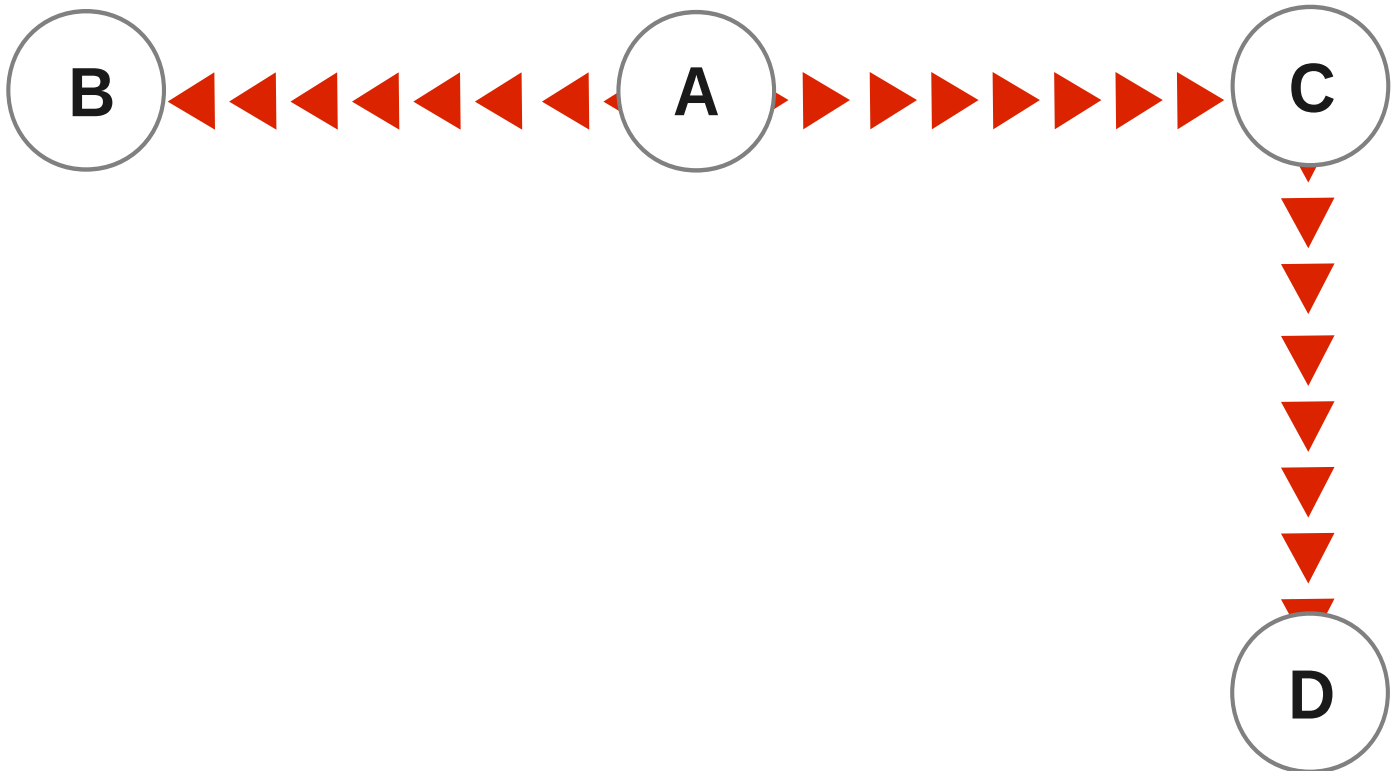
BC replication is intended for mission-critical applications that must be available 24 hours a day, 365 days a year, in the face of both unplanned events (such as system crashes) as well planned events (such as system and software upgrades).

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.

With SI replication, an instance has the ability to receive replication stream from another instance and at the same time execute its own business logic to compute and commit its own updates. In the following B←A→P→Q replication configuration, instance P can execute its own business logic to compute and commit its updates to its own replicated regions while receiving a replication stream from A. A is an originating primary instance, P is a supplementary originating instance, B is a secondary (standby) instance for A and Q is a secondary (standby) supplementary instance for P. In this configuration, B and Q are only permitted to receive and apply replication streams from their respective originating instances. Unlike P, B and Q cannot generate their own updates.
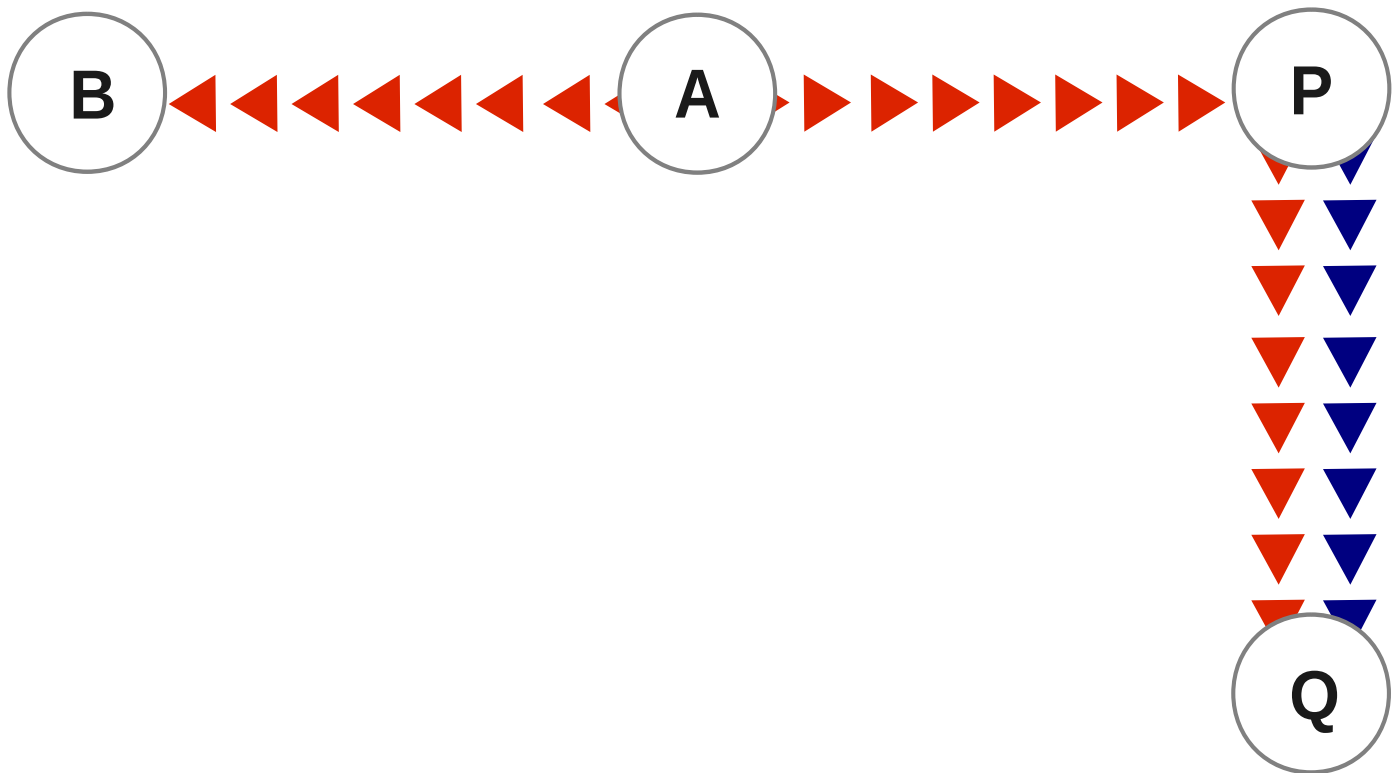
SI replication is a general purpose mechanism whose utility includes applications such as real-time decision support, warehousing, analytics, and auditing.

> **Note**
>
> In this book, instances {A, B, C...} denote instances setup for use as business continuity instances and instances {P, Q, R...} are set up for use as supplementary instances. An instance is a supplementary instance if its replication instance file is created with the -supplementary qualifier.

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.

Database Replication

Using TCP connections, GT.M replicates between instances with heterogeneous stacks of hardware, operating system, endian architecture 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 or later. 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 systems - 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 containing 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[1]. 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.

GT.M replication is asynchronous, which in turn means that the source and receiver ends of a replication connection are at an identical state 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. All updates in a transaction (within the TSTART/TCOMMIT boundary) have the same journal sequence number. Each update outside a transaction boundary has a unique journal sequence number.

On instances that do not include updates from supplementary logic, 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 in A. If P needs to roll {101,1,18} off its database in order to resynchronize replication with A, 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 journal sequence number on A becomes the stream sequence number on P for transactions replicated from A to P. In the example, the transaction that has the P sequence number of 101 has the sequence number 18 on A and B. 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 P has rolled off its transaction 18, and can include that when catching P up.

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.

---

[1]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.

LMS Group

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 to instance Q in Q's SI Group.

```
+----------------+
|      + A +     |
|      | | |     |
|B ----| | |---- P|
+--------|--------+
         |
         |
+--------V--------+
|      + Q +     |
|      |   |     |
|R ----|   |---- S|
+----------------+
```
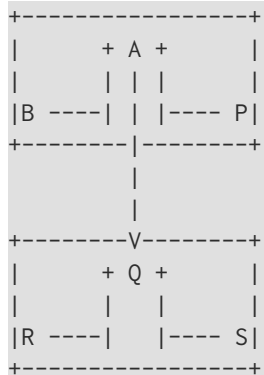
> 📌 **Note**
>
> In this replication configuration, instance B can also replicate to instance Q. However, instance P cannot replicate to an instance in Q's group because it is an SI member of A's BC group.

Database 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 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 numbers 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.

Instance Bound Global Directories

GDE provides a mapping option (GDE CHANGE -INSTANCE -FILE_NAME=<replication_instance_file>) to bind a global directory with a replication instance file. With this mapping option, GT.M processes give precedence to the <replication_instance_file> stored in GDE over any (or no) value of the gtm_repl_instance environment variable. This makes it possible for a GT.M process to update globals in the replicated regions of a different replication instance than that defined by the gtm_repl_instance environment variable.

Without this mapping option (that is, GDE CHANGE -INSTANCE -FILE_NAME= ""), an attempt to update a global bound to an instance other than the default replication instance file specified with the environment variable gtm_repl_instance produces the REPLINSTMISMTCH error:

```
GTM>set ^|"/path/to/unbound/XXXX.gld"|replnamespace="Hello from same system instance YYYY"
%GTM-E-REPLINSTMISMTCH, Process has replication instance file YYYY.repl (jnlpool shmid = 9999) open but database
▶ XXXX.dat is bound to instance file XXXX.repl (jnlpool shmid = 8888)
```

The following restrictions apply for making updates to the replicated regions of an instance bound global directory:

1. A replication instance file cannot share any region with another instance file.

2. The Source Servers of all the instances have properly set up Replication Journal Pools.

3. A TP transaction or a trigger, as it always executes within a TP transaction, must always restrict updates to globals in one replicating instance.

> **Note**
>
> - Processes other than MUPIP CREATE only use mapping information in a global directory, including collation, instance file association and name mapping, and ignore other (non-mapping) global directory database characteristics.
>
> - When Instance Freeze is enabled (gtm_custom_errors is appropriately defined), a process attaches a region to an instance at the first access to the region; the access may be a read or a VIEW/$VIEW(). Otherwise, the process attaches to a region at the first update to that region. When the mappings are correct, this difference does not matter.
>
> - A GT.M process can always update globals that are not in a replicated region.
>
> - Use $VIEW("JNLPOOL") to determine the state of the current Journal Pool. $VIEW("JNLPOOL") returns the replication instance file name for the current Journal Pool and an empty string when there is no Journal Pool. Note that the current Journal Pool may not be associated with the last global accessed by an extended reference.

> **Important**
>
> We believe that this functionality may interest both application developers and database administrators. Because this functionality has a wide variety of creative use cases when combined with other GT.M features like Triggers, TP, serialization with LOCK controls, replication filters, etc., we encourage GT.M users to explore this feature and provide us with feedback.

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).

Ardmore	BrynMawr	Malvern	Comments
O: ... *A95*, *A96*, *A97*, *A98*, *A99*	R: ... *A95*, *A96*, *A97*, *A98*	S: ... *M34*, *A95*, *M35*, *M36*, *A96*, *A97*, *M37*, *M38*	*Ardmore* as an originating primary instance at transaction number *A99*, replicates to *BrynMawr* as a BC replicating secondary instance at transaction number *A98* and *Malvern* as a SI that includes transaction number *A97*, interspersed with locally generated updates. Updates are recorded in each instance's journal files using before-image journaling.
Crashes	O: ... *A95*, *A96*, *A97*, *A98*, *B61*	... *M34*, *A95*, *M35*, *M36*, *A96*, *A97*, *M37*, *M38*	When an event disables *Ardmore*, *BrynMawr* becomes the new originating primary, with *A98* as the latest transaction in its database, and starts processing application logic to maintain business continuity. In this case where *Malvern* is not ahead of *BrynMawr*, the Receiver Server at *Malvern* can remain up after *Ardmore* crashes. When *BrynMawr* connects, its Source Server and *Malvern*'s Receiver Server confirms that *BrynMawr* is not behind *Malvern* with respect to updates received from *Ardmore*, and SI replication from *BrynMawr* picks up where replication from *Ardmore* left off.
-	O: ... *A95*, *A96*, *A97*, *A98*, *B61*, *B62*	S: ... *M34*, *A95*, *M35*, *M36*, *A96*, *A97*, *M37*, *M38*, *A98*, *M39*, *B61*, *M40*	*Malvern* operating as a supplementary instance to *BrynMawr* replicates transactions processed on *BrynMawr*, and also applies its own locally generated updates. Although *A98* was originally generated on *Ardmore*, *Malvern* received it from *BrynMawr* because *A97* was the common point between *BrynMawr* and *Malvern*.
... *A95*, *A96*, *A97*, *A98*, *A99*	O: ... *A95*, *A96*, *A97*, *A98*, *B61*, *B62*, *B63*, *B64*	S: ... *M34*, *A95*, *M35*, *M36*, *A96*, *A97*, *M37*, *M38*, *A98*, *M39*, *B61*, *M40*, *B62*, *B63*	*Malvern*, continuing as a supplementary instance to *BrynMawr*, replicates transactions processed on *BrynMawr*, and also applies its own locally generated updates. *Ardmore* meanwhile has been repaired and brought online. It has to roll transaction *A99* off its database into an Unreplicated Transaction Log before it can start operating as a replicating secondary instance to *BrynMawr*.
R: ... *A95*, *A96*, *A97*, *A98*, *B61*, *B62*, *B63*, *B64*	O: ... *A95*, *A96*, *A97*, *A98*, *B61*, *B62*, *B63*, *B64*, B65	S: ... *M34*, *A95*, *M35*, *M36*, *A96*, *A97*, *M37*, *M38*, *A98*, *M39*, *B61*, *M40*, *B62*, *B63*, *M41*, *B64*	Having rolled off transactions into an Unreplicated Transaction Log, *Ardmore* can now operate as a replicating secondary instance to *BrynMawr*. This is normal BC Logical Multi-Site operation. *BrynMawr* and *Malvern* 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: ... *A95*, *A96*, *A97*, *A98*, *A99*	R: ... *A95*, *A96*, *A97*	S: ... *M34*, *A95*, *M35*, *M36*, *A96*, *A97*, *M37*, *M38*, *A98*, *M39*, *M40*	*Ardmore* as an originating primary instance at transaction number *A99*, replicates to *BrynMawr* as a BC replicating secondary instance at transaction number *A97* and *Malvern* as a SI that includes transaction number *A98*, interspersed with locally generated updates. Updates are recorded in each instance's journal files using before-image journaling.

Ardmore	BrynMawr	Malvern	Comments
Crashes	O: ... *A95*, *A96*, *A97*	... *M34*, *A95*, *M35*, *M36*, *A96*, *A97*, *M37*, *M38*, *A98*, *M39*, *M40*	When an event disables **Ardmore**, **BrynMawr** becomes the new originating primary, with *A97* the latest transaction in its database. **Malvern** cannot immediately start replicating from **BrynMawr** because the database states would not be consistent - while **BrynMawr** does not have *A98* in its database and its next update may implicitly or explicitly depend on that absence, **Malvern** does, and may have relied on *A98* to compute *M39* and *M40*.
-	O: ... *A95*, *A96*, *A97*, *B61*, *B62*	S: ... *M34*, *A95*, *M35*, *M36*, *A96*, *A97*, *M37*, *M38*, *B61*	For **Malvern** to accept replication from **BrynMawr**, it must roll off transactions generated by **Ardmore**, (in this case *A98*) that **BrynMawr** does not have in its database, as well as any additional transactions generated and applied locally since transaction number *A98* from **Ardmore**.[a] This rollback is accomplished with a MUPIP JOURNAL -ROLLBACK -FETCHRESYNC operation on **Malvern**.[b] These rolled off transactions (*A98*, *M39*, *M40*) go into the Unreplicated Transaction Log and can be subsequently reprocessed by application code.[c] Once the rollback is completed, **Malvern** can start accepting replication from **BrynMawr**.[d] **BrynMawr** in its Originating Primary role processes transactions and provides business continuity, resulting in transactions *B61* and *B62*.
-	O: ... *A95*, *A96*, *A97*, *B61*, *B62*, *B63*, *B64*	S: ... *M34*, *A95*, *M35*, *M36*, *A96*, *A97*, *M37*, *M38*, *B61*, *B62*, *M39*a, *M40*a, *B63*	**Malvern** operating as a supplementary instance to **BrynMawr** replicates transactions processed on **BrynMawr**, and also applies its own locally generated updates. Note that *M39*a & *M40*a may or may not be the same updates as the *M39* & *M40* previously rolled off the database.

[a]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.

[b]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.

[c]GT.M's responsibility for them ends once it places them in the Unreplicated Transaction Log.

[d]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	BrynMawr	Malvern	Comments
O: ... *A95*, *A96*, *A97*, *A98*, *A99*	R: ... *A95*, *A96*, *A97*	S: ... *M34*, *A95*, *M35*, *M36*, *A96*, *A97*, *M37*, *M38*, *A98*, *M39*, *M40*	**Ardmore** as an originating primary instance at transaction number *A99*, replicates to **BrynMawr** as a BC replicating secondary instance at transaction number *A97* and **Malvern** as a SI that includes transaction number *A98*, interspersed with locally generated updates. Updates are recorded in each instance's journal files using before-image journaling.

Ardmore	BrynMawr	Malvern	Comments
Crashes	O: ... *A95*, *A96*, *A97*, *B61*, *B62*	... *M34*, *A95*, *M35*, *M36*, *A96*, *A97*, *M37*, *M38*, *A98*, *M39*, *M40*	When an event disables *Ardmore*, *BrynMawr* becomes the new originating primary, with *A97* the latest transaction in its database and starts processing application logic. Unlike the previous example, in this case, application design permits (or requires) *Malvern* to start replicating from *BrynMawr* even though *BrynMawr* does not have *A98* in its database and *Malvern* may have relied on *A98* to compute *M39* and *M40*.
-	O: ... *A95*, *A96*, *A97*, *B61*, *B62*	S: ... *M34*, *A95*, *M35*, *M36*, *A96*, *A97*, *M37*, *M38*, *A98*, *M39*, *M40*, *B61*, *B62*	With its Receiver Server started with the -noresync option, *Malvern* can receive a SI replication stream from *BrynMawr*, and replication starts from the last common transaction shared by *BrynMawr* and *Malvern*. Notice that on *BrynMawr* no *A98* precedes *B61*, whereas it does on *Malvern*, i.e., *Malvern* was ahead of *BrynMawr* with respect to the updates generated by *Ardmore*.

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	BrynMawr	Malvern	Newtown	Comments
O: ... *A95*, *A96*, *A97*, *A98*, *A99*	R: ... *A95*, *A96*, *A97*, *A98*	S: ... *M34*, *A95*, *M35*, *M36*, *A96*, *M37*, *A97*, *M38*	R: ... *M34*, *A95*, *M35*, *M36*, *A96*, *M37*	*Ardmore* as an originating primary instance at transaction number *A99*, replicates to *BrynMawr* as a BC replicating secondary instance at transaction number *A98* and *Malvern* as a SI that includes transaction number *A97*, interspersed with locally generated updates. *Malvern* in turn replicates to *Newtown*.
Goes down with the data center	O: ... *A95*, *A96*, *A97*, *A98*, *B61*, *B62*	Goes down with the data center	... *M34*, *A95*, *M35*, *M36*, *A96*, *M37*	When a data center outage disables *Ardmore*, and *Malvern*, *BrynMawr* becomes the new originating primary, with *A98* as the latest transaction in its database and starts processing application logic to maintain business continuity. *Newtown* can receive the SI replication stream from *BrynMawr*, without requiring a rollback since the receiver is not ahead of the source.
-	O: ... *A95*, *A96*, *A97*, *A98*, *B61*, *B62*	-	S: ... *M34*, *A95*, *M35*, *M36*, *A96*, *M37*, *A97*, *A98*, *N73*, *B61*, *N74*, *B62*	*Newtown* receives SI replication from *BrynMawr* and also applies its own locally generated updates. Although *A97* and *A98* were originally generated on *Ardmore*, *Newtown* receives them from *BrynMawr*. *Newtown* also computes and applies locally generated updates
... *A95*, *A96*, *A97*, *A98*, *A99*	O: ... *A95*, *A96*, *A97*, *B61*, *B62*, *B63*, *B64*	... *M34*, *A95*, *M35*, *M36*, *A96*, *M37*, *A97*, *M38*	S: ... *M34*, *A95*, *M35*, *M36*, *A96*, *M37*, *A97*, *A98*, *N73*, *B61*,	While *BrynMawr* and *Newtown*, keep the enterprise in operation, the first data center is recovered. Since *Ardmore* has

Ardmore	BrynMawr	Malvern	Newtown	Comments
			N74, **B62**, **N75**, **B63**, **N76**, **B64**	transactions in its database that were not replicated to **BrynMawr** when the latter started operating as the originating primary instance, and since **Malvern** had transactions that were not replicated to **Newtown** when the latter took over, **Ardmore** and **Malvern** must now rollback their databases and create Unreplicated Transaction Files before receiving BC replication streams from **BrynMawr** and **Newtown** respectively. **Ardmore** rolls off **A98** and **A99**, **Malvern** rolls off **A97** and **M38**.
R: ... **A95**, **A96**, **A97**, **B61**, **B62**, **B63**, **B64**	O: ... **A95**, **A96**, **A97**, **B61**, **B62**, **B63**, **B64**, B65	R: ... **M34**, **A95**, **M35**, **M36**, **A96**, **M37**, **A97**, **A98**, **N73**, **B61**, **N74**, **B62**, **N75**, **B63**, **N76**, **B64**	S: ... **M34**, **A95**, **M35**, **M36**, **A96**, **M37**, **A97**, **A98**, **N73**, **B61**, **N74**, **B62**, **N75**, **B63**, **N76**, **B64**, **N77**	Having rolled off transactions into an Unreplicated Transaction Log, **Ardmore** can now operate as a replicating secondary instance to **BrynMawr**. This is normal BC Logical Multi-Site operation. **BrynMawr** and **Malvern** continue operating as originating primary instance and supplementary instance. Note that having rolled **A97** off its database, **Malvern** receives that transaction from **Newtown** 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.

Ardmore	BrynMawr	Malvern	Comments
O: ... **A95**, **A96**, **A97**, **A98**, **A99**	R: ... **A95**, **A96**, **A97**	S: ... **M34**, **A95**, **M35**, **M36**, **A96**, **A97**, **M37**, **M38**, **A98**, **M39**, **M40**	**Ardmore** as an originating primary instance at transaction number **A99**, replicates to **BrynMawr** as a BC replicating secondary instance at transaction number **A97** and **Malvern** as a SI that includes transaction number **A98**, interspersed with locally generated updates. Updates are recorded in each instance's journal files using before-image journaling.
Rolls back to **A96** with **A97** through **A99** in the Unreplicated Transaction Log	Rolls back automatically to **A96** (assume Receiver Server started with -autorollback - refer to the V5.5-000 Release Notes for details.	-	Instances receiving a replication stream from **Ardmore** can be configured to rollback automatically when **Ardmore** performs an online rollback by starting the Receiver Server with -autorollback. If **Malvern**'s Receiver Server is so configured, it will roll **A97** through **M40** 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: ... **A95**, **A96**, **A97**a, **A98**a, **A99**a	R: ... **A95**, **A96**, **A97**a, **A98**a	S: ... **M34**, **A95**, **M35**, **M36**, **A96**, **A97**, **M37**, **M38**, **A98**,	Transactions **A97**a through **A99**a are different transactions from **A97** through **A99** (which are in an Unreplicated Transaction File on **Ardmore** and must be reprocessed). Note that **Malvern** has

Ardmore	BrynMawr	Malvern	Comments
		M39, M40, A97a, M41, A98a, M42	both the original A97 and A98 as well as A97a and A98a. A99 was never replicated to Malvern - Ardmore rolled back before it was replicated, and A99a has not yet made it to Malvern (it will soon, unless Ardmore rolls back again).

Limitations - SI Replication

starting V5.5-000, GT.M does not support replication between platforms with GT.M releases prior to V5.1-000. To upgrade to GT.M V5.5-000, first upgrade to GT.M V5.1-000 or later as an intermediate step.

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.
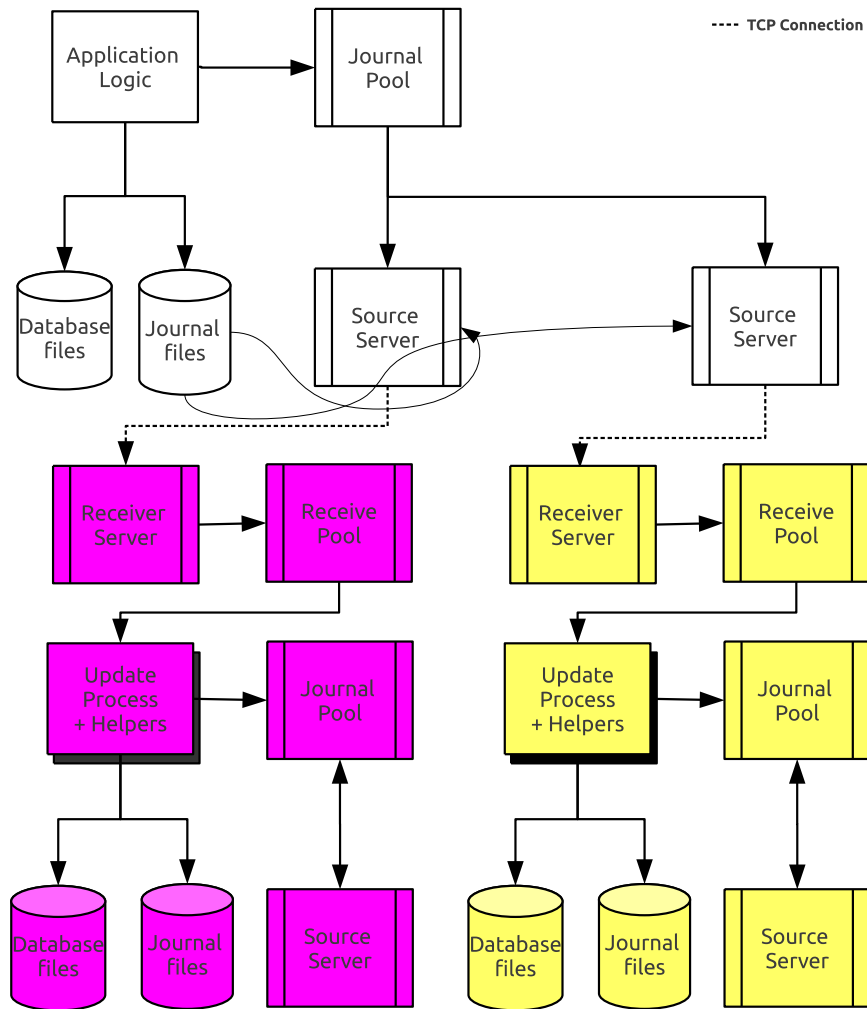
When a process commits a transaction on White, GT.M provides Durability by writing and "hardening" an update record to the journal file and then the database file. The same process also writes the update records to an area of shared memory called a Replication Journal Pool as part of committing the transaction, but does not wait for Rose and Yellow to commit the transaction

(this means 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 Replication 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 Replication Journal Pool to the Rose and Yellow Receiver Servers. The Replication 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 Replication 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 Replication Journal Pool. The diagram represents this with the curved lines from the journal file to the Source Server processes.

A Source Server 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 Replication 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.

On a primary instance, the first active Source Server creates the Replication Journal Pool before beginning to transmit updates to the secondary instance. Transition an active Source Server of a primary to passive when you want to pause replicating updates to the secondary instance but continue performing updates on the primary. When you are ready to resume replicating updates from the primary, transition the passive Source Server to active.

Replication requires each instance to have its own Replication Journal Pool. Additionally, an instance that receives a replication stream must have its own Receiver Pool.

On a secondary instance, the Receiver Server creates the Receiver Pool and a passive Source Server (pointing to either a replicating instance downstream or a dummy (non-existing) replicating instance when there is no replicating instance downstream) creates the Replication Journal Pool. White, Rose, and Yellow can be appropriate names of replication instances in the diagram. The passive Source Servers of Rose and Yellow instances both point to a dummy instance as there are no replicating instances downstream.

On Rose and Yellow instances, a Receiver Server receives update records sent by the White Source Server and puts them in the Receiver Pool, which is in a shared memory segment. Source and Receiver Server processes implement flow control to ensure that the Receiver Pool does not overflow. The Update Process picks these update records and writes them to the journal file, the database file, and the Replication Journal Pool. The Update Process on a replicating instance performs operations analogous to "Application Logic" on the originating 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 which makes GT.M less resource intensive than physical replication solutions. Furthermore, the Source Server can compress the byte stream which the Receiver Server then decompresses; alternatively network routers can perform the compression and decompression. When starting replication, you can enable TLS replication to secure replication between instances. Alternatively, you can use standard techniques at the stack or router for encrypting TCP connections to secure the replication stream.

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. They increase replication throughput, decrease backlog, and improve manageability.

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.

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, GT.M maintains the history of on instance as a set of records. GT.M adds new history records 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 MUPIP JOURNAL -ROLLBACK removes history records for rolled back updates from the tail of the instance file. 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, GT.M records the originating instance name as "Root Primary Instance Name" in the replication instance file history of both the instances. The same rule applies when a replicating instance acts as an originating instance for another replicating instance downstream.

This history serves to determine the journal sequence numbers through which both instances are synchronized when two instances attempt to connect. GT.M determines this journal sequence number by going back in the history of both the instance files to find the most recent shared journal sequence number generated by the 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 synchronization requires a MUPIP JOURNAL -ROLLBACK -FETCHRESYNC, or if configured, an autorollback on the Replicating Instance to rollback to a common synchronization point from which the originating instance can transmit updates to allow the Replicating Instance to catch up.

> **Note**
>
> Proper operation requires the Replication Instance file be consistent with the snapshot of the database files in a backup. MUPIP BACKUP -REPLINSTANCE creates a backup of the Replication Instance file. Before backing up the replication instance file, you must start the Source Server for the instance at least once . If the replication instance file is damaged or deleted, you must create a new instance file, and all recreate all downstream Replicating Instances 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 switchover 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.
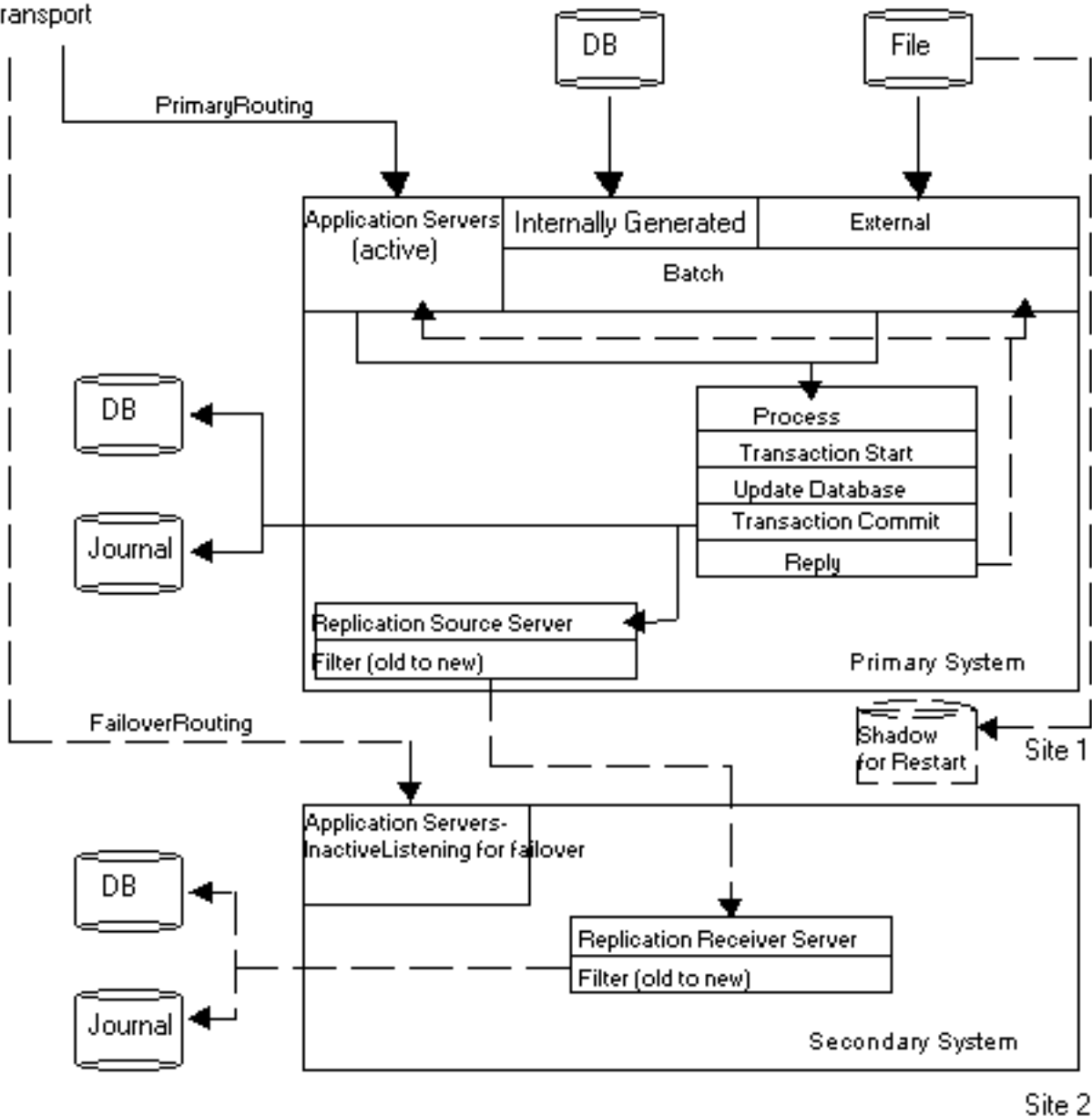
This diagram illustrates an application architecture that can reliably perform batch and online updates in a tiered environment. It addresses the online updates via the Message Transport (which has to be able to reroute communications to the current

originating instance after a switchover) and batch updates via an external file (which has to be made available to the current originating instance after a switchover).

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 switchover-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 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 switchover 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.

Switchover

Switchover 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 switchover is outside the scope of GT.M. FIS recommends you to adhere to the following rules while designing switchover:

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 switchover 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 switchover is a wholesome practice for maximizing business continuity. FIS strongly recommends setting up a switchover 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 switchover 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 254).

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.

MUPIP REPLIC -SOURCE -JNLPOOL -SHOW displays whether the custom errors file is loaded.

> **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.

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**
>
> 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 during replication. 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. Please refer to Section : "Setting up a secured TLS replication connection" (page 286) for an example for setting up replication between two instances and encrypting their replication connection using TLS replication. Note that this example uses a demo set of certificates.

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 switchover 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[2] 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 tbck 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 that better 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.

---

[2]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[a] 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	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. • 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[b] - until the next epoch. Before image journal records are larger than journal records that describe updates. • At epochs, both before image journaling and nobefore journaling flush journal blocks and perform an fsync() on journal files[c]. When using before image journaling, GT.M ensures all dirty database blocks have been written and does an fsync()[d], but nobefore journaling does not take these steps. 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.

[a]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.

[b]How much the steady level is lower depends on the application and workload.

[c]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.

[d]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

Download Replication Examples

repl_procedures.tar.gz contains a set of replication example scripts. Each script contains a combination of GT.M commands that accomplish a specific task. All examples in the Procedures section use these replication scripts but each example uses different script sequence and script arguments. Always run all replication examples in a test system from a new directory as they create sub-directories and database files in the current directory. No claim of copyright is made with regard to these examples. These example scripts are for explanatory purposes and are not intended for production use. YOU MUST UNDERSTAND, AND APPROPRIATELY ADJUST THE COMMANDS GIVEN IN THESE SCRIPTS BEFORE USING IN A PRODUCTION ENVIRONMENT. Typically, you would set replication between instances on different systems/data centers and

create your own set of scripts with appropriate debugging and error handling to manage replication between them. Click ▼ to download *repl_procedures.tar.gz* on a test system.

repl_procedures.tar.gz includes the following scripts:

env

Sets a default environment for GT.M replication. It take two arguments:

- The name of the instance/database directory.

- The GT.M version.

Example: **source ./env A V6.3-000A_x86_64**

Here is the code:

```
export gtm_dist=/usr/lib/fis-gtm/$2
export gtm_repl_instname=$1
export gtm_repl_instance=$PWD/$gtm_repl_instname/gtm.repl
export gtmgbldir=$PWD/$gtm_repl_instname/gtm.gld
export gtm_principal_editing=EDITING
export gtmroutines="$PWD/$gtm_repl_instname $gtm_dist"
#export gtmroutines="$PWD/$gtm_repl_instname $gtm_dist/libgtmutil.so"
# Here is an example of setting the gtmroutines environment variable:
# if [ -e  "$gtm_dist/libgtmutil.so" ] ; then export gtmroutines="$PWD/$gtm_repl_instname $gtm_dist/libgtmutil.so"
else export gtmroutines="$PWD/$gtm_repl_instname* $gtm_dist" ; fi
# For more examples on setting GT.M related environment variables to reasonable values on POSIX shells, refer to
 the
▶ gtmprofile script.
#export gtmcrypt_config=$PWD/$gtm_repl_instname/config_file
#echo -n "Enter Password for gtmtls_passwd_${gtm_repl_instname}: ";export
▶ gtmtls_passwd_${gtm_repl_instname}="`$gtm_dist/plugin/gtmcrypt/maskpass|tail -n 1|cut -f 3 -d " "`"
```

▼

Modify the env script according to your test environment.

db_create

Creates a new sub-directory in the current directory, a global directory file with settings from gdemsr, and the GT.M database file.

Here is the code:

```
mkdir -p $PWD/$gtm_repl_instname/
  $gtm_dist/mumps -r ^GDE @gdemsr
  $gtm_dist/mupip create
```

gdemsr contains:

```
change -segment DEFAULT -file_name=$PWD/$gtm_repl_instname/gtm.dat
  exit
```

backup_repl

Creates a backup of the replication instance file. The first argument specifies the location of the backed up replication instance file.

Here is the code:

```
$gtm_dist/mupip backup -replinst=$1
```

repl_setup

Turns on replication for all regions and create the replication instance file with the -noreplace qualifier for a BC instance.

Here is the code:

```
$gtm_dist/mupip set -replication=on -region "*"
  $gtm_dist/mupip replicate -instance_create -noreplace
```

originating_start

Starts the Source Server of the originating instance in a BC replication configuration. It takes five arguments:

- The first argument is the name of the originating instance. This argument is also used in the name of the Source Server log file.

- The second argument is the name of the BC replicating instance. This argument is also used in the name of the Source Server log file.

- The third argument is port number of localhost at which the Receiver Server is waiting for a connection.

- The optional fourth and fifth argument specify the -tlsid and -reneg qualifiers used to set up a TLS/SSL connection.

- Example:./**originating_start A B 4001**

Here is the code:

```
$gtm_dist/mupip replicate -source -start -instsecondary=$2 -secondary=localhost:$3 -buffsize=1048576 -log=$PWD/$1/$1_$2.log $4
 $5
tail -30 $PWD/$1/$1_$2.log
$gtm_dist/mupip replicate -source -checkhealth
```

replicating_start

Starts the passive Source Server and the Receiver Server in a BC replication configuration. It takes four arguments:

- The first argument is the name of the replicating instance. This argument is also used in the name of the passive Source Server and Receiver Server log file name.

- The second argument is port number of localhost at which the Source Server is sending the replication stream for the replicating instance.

- The optional third and fourth arguments are used to specify additional qualifiers for the Receiver Server startup command.

- Example:./**replicating_start B 4001**

Here is the code:

```
$gtm_dist/mupip replicate -source -start -passive -instsecondary=dummy -buffsize=1048576
 -log=$PWD/$1/source$1_dummy.log # creates the Journal Pool
$gtm_dist/mupip replicate -receive -start -listenport=$2 -buffsize=1048576 -log=$PWD/$1/receive.log $3 $4 # starts
 the
 Receiver Server
```

```
tail -20 $PWD/$1/receive.log
$gtm_dist/mupip replicate -receive -checkhealth
```

▼

suppl_setup

Turns on replication for all regions, create the supplementary replication instance file with the -noreplace qualifier, starts the passive Source Server, starts the Receiver Server of an SI replicating instance, and displays the health status of the Receiver Server and Update Process. Use this to start an SI replicating instance for the first time. It takes four arguments:

- The first argument is the name of the supplementary instance. This argument is also used in the name of the passive Source Server and Receiver Server log files.

- The second argument is the path to the backed up replication instance file of the originating instance.

- The third argument is port number of localhost at which the Receiver Server is waiting for a connection.

- The optional fourth argument is either -updok or -updnotok which determines whether the instance accepts updates.

- The optional fifth argument specifies -tlsid which is used in setting up a TLS/SSL replication connection.

Example: **./suppl_setup P startA 4011 -updok**

Here is the code:

```
$gtm_dist/mupip set -replication=on -region "*"
$gtm_dist/mupip replicate -instance_create -supplementary -noreplace
$gtm_dist/mupip replicate -source -start -passive -buf=1048576 -log=$PWD/$gtm_repl_instname/$1_dummy.log
▶ -instsecondary=dummy $4
$gtm_dist/mupip replicate -receive -start -listenport=$3 -buffsize=1048576 -log=$PWD/$gtm_repl_instname/$1.log
▶ -updateresync=$2 -initialize $5
tail -30 $PWD/$1/$1.log
$gtm_dist/mupip replicate -receive -checkhealth
```

▼

repl_status

Displays health and backlog status information for the Source Server and Receiver Server in the current environment.

Here is the code:

```
echo "--------------------------------------------------------------"
  echo "Source Server $gtm_repl_instname: "
  echo "--------------------------------------------------------------"
  $gtm_dist/mupip replicate -source -check
  $gtm_dist/mupip replicate -source -showbacklog
  echo "--------------------------------------------------------------"
  echo "Receiver Server $gtm_repl_instname: "
  echo "--------------------------------------------------------------"
  $gtm_dist/mupip replicate -receive -check
```

```
$gtm_dist/mupip replicate -rece -showbacklog
```

rollback

Performs an ONLINE FETCHRESYNC rollback and creates a lost and/or broken transaction file. It takes two arguments:

- The first argument specifies the communication port number that the rollback command uses when fetching the reference point. This is the same port number that the Receiver Server used to communicate with the Source Server.

- The second argument specifies either **backward** or **forward**.

Example: **./rollback 4001 backward**

Here is the code:

```
$gtm_dist/mupip journal -rollback -fetchresync=$1 -$2 "*"
```

originating_stop

Shuts down the Source Server with a two second timeout and perform a MUPIP RUNDOWN operation.

The first argument specifies additional qualifiers for the Source Server shutdown command.

Here is the code:

```
$gtm_dist/mupip replicate -source -shutdown -timeout=2 $1 #Shut down the originating Source Server
  $gtm_dist/mupip rundown -region "*" #Perform database rundown
```

replicating_stop

Shuts down the Receiver Server with a two seconds timeout and then shuts down the passive Source Server.

Here is the code:

```
$gtm_dist/mupip replicate -receiver -shutdown -timeout=2 #Shut down the Receiver Server
  $gtm_dist/mupip replicate -source -shutdown -timeout=2 #Shut down the passive Source Server
```

replicating_start_suppl_n

Starts the passive Source Server and the Receiver Server of the supplementary instance for all startups except the first. It takes three arguments:

- The first argument is the name of the supplementary instance. It is also used in the names of the passive Source Server and the Receiver Server log files.

- The second argument is port number of localhost at which the Receiver Server is waiting for a connection.

- The optional third argument is an additional qualifier for the passive Source Server startup command. In the examples, the third argument is either **-updok** or **-updnotok**.

- The optional fourth argument is an additional qualifier for the Receiver Server startup command. In the examples, the fourth argument is either **-autorollback** or **-noresync**.

- The optional fifth argument is -tlsid which is used to set up a TLS/SSL replication connection.

Example:**./replicating_start_suppl_n P 4011 -updok -noresync**

Here is the code:

```
$gtm_dist/mupip replicate -source -start -passive -instsecondary=dummy -buffsize=1048576
 -log=$PWD/$gtm_repl_instname/$12dummy.log $3 # creates the Journal Pool
  $gtm_dist/mupip replicate -receive -start -listenport=$2 -buffsize=1048576 $4 $5 -log=$PWD/$gtm_repl_instname/
$1.log
 # starts the Receiver Server and the Update Process
  tail -30 $PWD/$1/$1.log
  $gtm_dist/mupip replicate -receiver -checkhealth # Checks the health of the Receiver Server and the Update
 Process
```

Setting up an A→B replication configuration with empty databases

On A:

- Turn on replication.

- Create the replication instance file.

- Start the Source Server.

On B:

- Turn on replication.

- Create a new replication instance file.

- Start the passive Source Server.

- Start the Receiver Server.

Example:

```
source ./gtmenv A V6.3-000A_x86_64
./db_create
./repl_setup
./originating_start A B 4001
source ./gtmenv B V6.3-000A_x86_64
./db_create
./repl_setup
./replicating_start B 4001
./repl_status
```

The shutdown sequence is as follows:

```
source ./gtmenv B V6.3-000A_x86_64
./replicating_stop
source ./env A V6.3-000A_x86_64
./originating_stop
```

Setting up an A→B→C replication configuration with empty databases

On A:

- Turn on replication.

- Create the replication instance file.

- Start the Source Server.

On B:

- Turn on replication.

- Create a new replication instance file.

- Start the passive Source Server.

- Start the Receiver Server.

- Start the Source Server with the -propagateprimary qualifier.

On C:

- Turn on replication.

- Create a new replication instance file.

- Start the passive Source Server .

- Start the Receiver Server.

Example:

```
source ./gtmenv A V6.3-000A_x86_64
./db_create
./repl_setup
./originating_start A B 4001
source ./gtmenv B V6.3-000A_x86_64
./db_create
./repl_setup
./replicating_start B 4001
./originating_start B C 4002 -propagateprimary
source ./gtmenv C V6.3-000A_x86_64
./db_create
./repl_setup
./replicating_start C 4002
./repl_status
```

The shutdown sequence is as follows:

```
source ./gtmenv C V6.3-000A_x86_64
./replicating_stop
source ./gtmenv B V6.3-000A_x86_64
./replicating_stop
./originating_stop
```

```
source ./gtmenv A V6.3-000A_x86_64
./originating_stop
```

Setting up an A→P replication configuration with empty databases

On A:

- Turn on replication.

- Create a new replication instance file.

- Start the Source Server.

- Immediately, after starting the Source Server but before making any updates, take a backup of the replication instance file. The instance file has the journal sequence number that corresponds to the database state at the time of starting the instance. This backup instance helps when you need to start a new Supplementary Instance without taking a backup of the Originating Instance. Retain the backup copy of the Originating Instance as you may need it in future as a checkpoint from where its supplementary instance may resume to receive updates.

> **Note**
>
> Note that while a backed up instance file helps start replication on the P side of A→P, it does not prevent the need for taking a backup of the database on A. You need to do a database backup/restore or an extract/load from A to P to ensure P has all of the data as on A at startup.

On P:

- Turn on replication.

- Create a new 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 start ups, do not use -updateresync -initialize with qualifiers. Either use -autorollback with the Receiver Server startup command or perform an explicit -fetchresync -rollback before starting the Receiver Server.

Example:

```
source ./gtmenv A V6.3-000A_x86_64
./db_create
./repl_setup
./originating_start A P 4000
./backup_repl startA
source ./gtmenv P V6.3-000A_x86_64
./db_create
./suppl_setup P startA 4000 -updok
./repl_status
# For subsequent Receiver Server startup for P, use:
# ./replicating_start_suppl_n P 4000 -updok -autorollback
```

```
# or
#./rollback 4000 backward
#./replicating_start_suppl_n P 4000 -updok
```

The shutdown sequence is as follows:

```
source ./gtmenv P V6.3-000A_x86_64
./replicating_stop
source ./gtmenv A V6.3-000A_x86_64
./originating_stop
```

Replicating Instance Starts from Backup of Originating Instance (A→B and A→P)

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.

On A:

- Create a backup using -DATABASE, -REPLINST, -NEWJNLFILE=PREVLINK, and -BKUPDBJNL=DISABLE qualifiers. -DATABASE creates a comprehensive backup of the database file. -REPLINST backs up the replication instance file. -BKUPDBJNL=DISABLE scrubs all journal file information in the backup database file. As the backup of instance A is comprehensive, -NEWJNLFILE=PREVLINK cuts the back link to prior generation journal files of the database for which you are taking the backup.

- Copy the backup of the replication instance file to the location of the backed up instance.

- Start a new Source Server for the backed up replicating instance.

On the backed up 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.

- Run MUPIP REPLICATE -EDITINSTANCE command to change the name of the backed up replication instance file.

- Start the Receiver Server for the BC replicating instance. Do not use the -UPDATERESYNC qualifier to start the receiver server of a BC replicating instance. -UPDATERESYNC is necessary when you start the Receiver Server of a SI replicating instance for the first time. Without -UDPATERESYNC, the SI replicating instance may refuse to start replication because the journal sequence number in the replicating instance may be higher than what the originating instance expects.

Example:

The following example demonstrates starting a replicating instance from the backup of an originating instance in an A→B replication configuration. Note that you do not need to perform an -updateresync when starting a BC replicating instance for the first time.

```
source ./gtmenv A V6.3-000A_x86_64
./db_create
./repl_setup
./originating_start A backupA 4001
./backup_repl startingA   #Preserve the backup of the replicating instance file that represents the state at the
 time
▶ of starting the instance.
```

```
$gtm_dist/mumps -r %XCMD 'for i=1:1:10 set ^A(i)=i'
mkdir backupA
$gtm_dist/mupip backup -replinst=currentstateA -newjnlfile=prevlink -bkupdbjnl=disable DEFAULT backupA
source ./gtmenv backupA V6.3-000A_x86_64
./db_create
./repl_setup
cp currentstateA backupA/gtm.repl
$gtm_dist/mupip replicate -editinstance -name=backupA backupA/gtm.repl
./replicating_start backupA 4001
./repl_status
```

The shutdown sequence is as follows:

```
source ./gtmenv backupA V6.3-000A_x86_64
./replicating_stop
source ./gtmenv A V6.3-000A_x86_64
./originating_stop
```

The following example demonstrates starting a replicating instance from the backup of an originating instance in an A→P replication configuration. Note that you need to perform an -updateresync to start a supplementary instance for the first time.

```
source ./gtmenv A V6.3-000A_x86_64
./db_create
./repl_setup
./originating_start A backupA 4011
./backup_repl startingA
$gtm_dist/mumps -r %XCMD 'for i=1:1:10 set ^A(i)=i'
./backup_repl currentstateA
mkdir backupA
$gtm_dist/mupip backup -newjnlfile=prevlink -bkupdbjnl=disable DEFAULT backupA
source ./gtmenv backupA V6.3-000A_x86_64
./db_create
./suppl_setup backupA currentstateA 4011 -updok
./repl_status
```

The shutdown sequence is as follows:

```
source ./gtmenv backupA V6.3-000A_x86_64
./replicating_stop
source ./gtmenv A V6.3-000A_x86_64
./originating_stop
```

Switchover possibilities in an A→B replication configuration

A switchover is the procedure of switching the roles of an originating instance and a replicating instance. A switchover is necessary for various reasons including (but not limited to) testing the replicating instance preparedness to take over the role of an originating instance or bringing the originating instance down for maintenance in a way that there is minimal impact on application availability.

In an A->B replication configuration, at any given point there can be two possibilities:

• A is ahead of B, that is A has updates which are not yet replicated to B.

- A and B are in sync. This happens where there are no new updates on A and all pending updates are replicated to B.

The steps described in this section perform a switchover (A->B becomes B->A) under both these possibilities. When A is ahead of B, these steps generate a lost transaction file which must be applied to the new originating instance as soon as possible. The lost transaction file contains transactions which are were not replicated to B. 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(), perform two additional steps (mupip replicate -source -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.

First, choose a time when there are no database updates or the rate of updates are low to minimize the chances that your application may time out. There may be a need to hold database updates briefly during the switchover. For more information on holding database updates, refer to Instance Freeze section to configure an appropriate freezing mechanism suitable for your environment.

In an A→B replication configuration, follow these steps:

On A:

- 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 30 seconds.

On B:

1. Shut down the Receiver Server and the Update Process.

2. Shut down the passive Source Server to bring down the journal pool. Ensure that you shut down the Receiver Server and Update Process first before shutting down the passive Source Server.

3. Start B as the new originating instance.

On A:

1. Start the passive Source Server.

2. Perform a FETCHRESYNC ROLLBACK BACKWARD.

3. Start the Receiver Server.

4. Process the lost transaction file as soon as possible.

The following example runs a switchover in an A→B replication configuration.

```
source ./gtmenv A V6.3-000A_x86_64 # creates a simple environment for instance A
./db_create
./repl_setup # enables replication and creates the replication instance file
./originating_start A B 4001 # starts the active Source Server (A->B)
$gtm_dist/mumps -r %XCMD 'for i=1:1:100 set ^A(i)=i'
./repl_status #-SHOWBACKLOG and -CHECKHEALTH report
source ./gtmenv B V6.3-000A_x86_64 # creates a simple environment for instance B
./db_create
./repl_setup
./replicating_start B 4001
```

```
./repl_status # -SHOWBACKLOG and -CHECKHEATH report
./replicating_stop # Shutdown the Receiver Server and the Update Process
source ./gtmenv A V6.3-000A_x86_64 # Creates an environment for A
$gtm_dist/mumps -r %XCMD 'for i=1:1:50 set ^losttrans(i)=i' # perform some updates when replicating instance is
 not
▶ available.
sleep 2
./originating_stop # Stops the active Source Server
source ./gtmenv B V6.3-000A_x86_64 # Create an environment for B
./originating_start B A 4001 # Start the active Source Server (B->A)
source ./gtmenv A V6.3-000A_x86_64 # Create an environment for A
./rollback 4001 backward
./replicating_start A 4001 # Start the replication Source Server
./repl_status # To confirm whether the Receiver Server and the Update Process started correctly.
cat A/gtm.lost
```

▼

The shutdown sequence is as follows:

```
source ./gtmenv A V6.3-000A_x86_64
./replicating_stop
source ./gtmenv B V6.3-000A_x86_64
./originating_stop
```

Switchover possibilities in a B←A→P replication configuration

A requires rollback

The following scenario demonstrates a switchover from B←A→P to A←B→P when A has unreplicated updates that require rollback before B can become the new originating instance.

A	B	P	Comments
O: ... *A95*, *A96*, *A97*, *A98*, *A99*	R: ... *A95*, *A96*, *A97*, *A98*	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *A97*, *P37*, *P38*	*A* as an originating primary instance at transaction number *A99*, replicates to *B* as a BC replicating secondary instance at transaction number *A98* and *P* as a SI that includes transaction number *A97*, interspersed with locally generated updates. Updates are recorded in each instance's journal files using before-image journaling.
Crashes	O: ... *A95*, *A96*, *A97*, *A98*, *B61*	... *P34*, *A95*, *P35*, *P36*, *A96*, *A97*, *P37*, *P38*	When an event disables *A*, *B* becomes the new originating primary, with *A98* as the latest transaction in its database, and starts processing application logic to maintain business continuity. In this case where *P* is not ahead of *B*, the Receiver Server at *P* can remain up after *A* crashes. When *B* connects, its Source Server and *P*'s Receiver Server confirms that *B* is not behind *P* with respect to updates received from *A*, and SI replication from *B* picks up where replication from *A* left off.
-	O: ... *A95*, *A96*, *A97*, *A98*, *B61*, *B62*	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *A97*, *P37*, *P38*, *A98*, *P39*, *B61*, *P40*	*P* operating as a supplementary instance to *B* replicates transactions processed on *B*, and also applies its own locally generated updates. Although *A98* was originally generated on *A*, *P* received it from *B* because *A97* was the common point between *B* and *P*.

A	B	P	Comments
... *A95*, *A96*, *A97*, *A98*, *A99*	O: ... *A95*, *A96*, *A97*, *A98*, *B61*, *B62*, *B63*, *B64*	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *A97*, *P37*, *P38*, *A98*, *P39*, *B61*, *P40*, *B62*, *B63*	*P*, continuing as a supplementary instance to *B*, replicates transactions processed on *B*, and also applies its own locally generated updates. *A* meanwhile has been repaired and brought online. It has to roll transaction *A99* off its database into an Unreplicated Transaction Log before it can start operating as a replicating secondary instance to *B*.
R: ... *A95*, *A96*, *A97*, *A98*, *B61*, *B62*, *B63*, *B64*	O: ... *A95*, *A96*, *A97*, *A98*, *B61*, *B62*, *B63*, *B64*, B65	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *A97*, *P37*, *P38*, *A98*, *P39*, *B61*, *P40*, *B62*, *B63*, *P41*, *B64*	Having rolled off transactions into an Unreplicated Transaction Log, *A* can now operate as a replicating secondary instance to *B*. This is normal BC Logical Multi-Site operation. *B* and *P* continue operating as originating primary instance and supplementary instance.

The following example creates this switchover scenario:

```
source ./gtmenv A V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'set ^A(98)=99'
source ./gtmenv B V6.3-000A_x86_64
./replicating_stop
source ./gtmenv A V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'set ^A(99)=100'
./originating_stop
source ./gtmenv B V6.3-000A_x86_64
./originating_start B A 4010
./originating_start B P 4011
./backup_repl startB
$gtm_dist/mumps -r ^%XCMD 'set ^B(61)=0'
source ./gtmenv P V6.3-000A_x86_64
./suppl_setup M startB 4011 -updok
$gtm_dist/mumps -r ^%XCMD 'for i=39:1:40 set ^P(i)=i'
source ./gtmenv B V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'set ^B(62)=1,^B(63)=1'
source ./gtmenv A V6.3-000A_x86_64
./rollback 4010 backward
./replicating_start A 4010
source ./gtmenv B V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'set ^B(64)=1,^B(65)=1'
cat A/gtm.lost
```

The shutdown sequence is as follows:

```
source ./gtmenv B V6.3-000A_x86_64
./originating_stop
source ./gtmenv A V6.3-000A_x86_64
./replicating_stop
source ./gtmenv P V6.3-000A_x86_64
./replicating_stop
```

A and P require rollback

The following demonstrates a switchover scenario from B←A→P to A←B→P where A and P have unreplicated updates that require rollback before B can become the new originating instance.

A	B	P	Comments
O: ... *A95*, *A96*, *A97*, *A98*, *A99*	R: ... *A95*, *A96*, *A97*	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *A97*, *P37*, *P38*, *A98*, *P39*, *P40*	*A* as an originating primary instance at transaction number *A99*, replicates to *B* as a BC replicating secondary instance at transaction number *A97* and *P* as a SI that includes transaction number *A98*, interspersed with locally generated updates. Updates are recorded in each instance's journal files using before-image journaling.
Crashes	O: ... *A95*, *A96*, *A97*	... *P34*, *A95*, *P35*, *P36*, *A96*, *A97*, *P37*, *P38*, *A98*, *P39*, *P40*	When an event disables *A*, *B* becomes the new originating primary, with *A97* the latest transaction in its database. *P* cannot immediately start replicating from *B* because the database states would not be consistent - while *B* does not have *A98* in its database and its next update may implicitly or explicitly depend on that absence, *P* does, and may have relied on *A98* to compute *P39* and *P40*.
-	O: ... *A95*, *A96*, *A97*, *B61*, *B62*	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *A97*, *P37*, *P38*, *B61*	For *P* to accept replication from *B*, it must roll off transactions generated by *A*, (in this case *A98*) that *B* does not have in its database, as well as any additional transactions generated and applied locally since transaction number *A98* from *A*.[a] This rollback is accomplished with a MUPIP JOURNAL -ROLLBACK -FETCHRESYNC operation on *P*.[b] These rolled off transactions (*A98*, *P39*, *P40*) go into the Unreplicated Transaction Log and can be subsequently reprocessed by application code.[c] Once the rollback is completed, *P* can start accepting replication from *B*.[d] *B* in its Originating Primary role processes transactions and provides business continuity, resulting in transactions *B61* and *B62*.
-	O: ... *A95*, *A96*, *A97*, *B61*, *B62*, *B63*, *B64*	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *A97*, *P37*, *P38*, *B61*, *B62*, *P39*a, *P40*a, *B63*	*P* operating as a supplementary instance to *B* replicates transactions processed on *B*, and also applies its own locally generated updates. Note that *P39*a & *P40*a may or may not be the same updates as the *P39* & *P40* previously rolled off the database.

[a]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.

[b]In scripting for automating operations, there is no need to explicitly test whether *B* is behind *P* - 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 *P* followed by a reconnection attempt by *B*. On the other hand, there is no harm in *P* routinely performing a rollback before having *B* 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.

[c]GT.M's responsibility for them ends once it places them in the Unreplicated Transaction Log.

[d]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.

The following example creates this scenario.

```
source ./gtmenv A V6.3-000A_x86_64
./db_create
./repl_setup
./originating_start A B 4010
./originating_start A P 4011
./backup_repl startA
$gtm_dist/mumps -r ^%XCMD 'for i=1:1:97 set ^A(i)=i'
source ./gtmenv B V6.3-000A_x86_64
./db_create
./repl_setup
./replicating_start B 4010
source ./gtmenv P V6.3-000A_x86_64
```

```
./db_create
./suppl_setup P startA 4011 -updok
$gtm_dist/mumps -r ^%XCMD 'for i=1:1:40 set ^P(i)=i'
source ./gtmenv B V6.3-000A_x86_64
./replicating_stop
source ./gtmenv A V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'set ^A(98)=99'
source ./gtmenv P V6.3-000A_x86_64
./replicating_stop
source ./gtmenv A V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'set ^A(99)=100'
./originating_stop
source ./gtmenv B V6.3-000A_x86_64
./originating_start B A 4010
./originating_start B P 4011
./backup_repl startB
$gtm_dist/mumps -r ^%XCMD 'set ^B(61)=0,^B(62)=1'
source ./gtmenv P V6.3-000A_x86_64
./rollback 4011 backward
./suppl_setup P startB 4011 -updok
$gtm_dist/mumps -r ^%XCMD 'for i=39:1:40 set ^P(i)=i'
source ./gtmenv A V6.3-000A_x86_64
./rollback 4010 backward
./replicating_start A 4010
source ./gtmenv B V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'set ^B(64)=1,^B(65)=1'
cat A/gtm.lost
cat P/gtm.lost
```

The shutdown sequence is as follows:

```
source ./gtmenv B V6.3-000A_x86_64
./originating_stop
source ./gtmenv A V6.3-000A_x86_64
./replicating_stop
source ./gtmenv P V6.3-000A_x86_64
./replicating_stop
```

Rollback not required by application design

The following scenario demonstrates a switchover from B←A→P to A←B→P when A and P have unreplicated updates. By application design, unreplicated updates on P do not require rollback when B becomes the new originating instance.

A	B	P	Comments
O: ... *A95*, *A96*, *A97*, *A98*, *A99*	R: ... *A95*, *A96*, *A97*	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *A97*, *P37*, *P38*, *A98*, *P39*, *P40*	*A* as an originating primary instance at transaction number *A99*, replicates to *B* as a BC replicating secondary instance at transaction number *A97* and *P* as a SI that includes transaction number *A98*, interspersed with locally generated updates. Updates are recorded in each instance's journal files using before-image journaling.
Crashes	O: ... *A95*, *A96*, *A97*, *B61*, *B62*	... *P34*, *A95*, *P35*, *P36*, *A96*, *A97*, *P37*, *P38*, *A98*, *P39*, *P40*	When an event disables *A*, *B* becomes the new originating primary, with *A97* the latest transaction in its database and starts processing application logic. Unlike the previous example, in this case, application design permits (or requires) *P* to start replicating from

A	B	P	Comments
			B even though *B* does not have *A98* in its database and *P* may have relied on *A98* to compute *P39* and *P40*.
-	O: ... *A95*, *A96*, *A97*, *B61*, *B62*	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *A97*, *P37*, *P38*, *A98*, *P39*, *P40*, *B61*, *B62*	With its Receiver Server started with the -noresync option, *P* can receive a SI replication stream from *B*, and replication starts from the last common transaction shared by *B* and *P*. Notice that on *B* no *A98* precedes *B61*, whereas it does on *P*, i.e., *P* was ahead of *B* with respect to the updates generated by *A*.

The following example creates this scenario.

```
source ./gtmenv A V6.3-000A_x86_64
./db_create
./repl_setup
./originating_start A B 4010
./originating_start A P 4011
./backup_repl startA
$gtm_dist/mumps -r ^%XCMD 'for i=1:1:97 set ^A(i)=i'
source ./gtmenv B V6.3-000A_x86_64
./db_create
./repl_setup
./replicating_start B 4010
source ./gtmenv P V6.3-000A_x86_64
./db_create
./suppl_setup P startA 4011 -updok
$gtm_dist/mumps -r ^%XCMD 'for i=1:1:40 set ^P(i)=i'
source ./gtmenv B V6.3-000A_x86_64
./replicating_stop
source ./gtmenv A V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'set ^A(98)=99'
source ./gtmenv P V6.3-000A_x86_64
./replicating_stop
source ./gtmenv A V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'set ^A(99)=100'
./originating_stop
source ./gtmenv B V6.3-000A_x86_64
./originating_start B A 4010
./originating_start B P 4011
#./backup_repl startB
$gtm_dist/mumps -r ^%XCMD 'set ^B(61)=0,^B(62)=1'
source ./gtmenv P V6.3-000A_x86_64
./replicating_start_suppl_n P 4011 -updok -noresync
$gtm_dist/mumps -r ^%XCMD 'for i=39:1:40 set ^P(i)=i'
source ./gtmenv A V6.3-000A_x86_64
./rollback 4010 backward
./replicating_start A 4010
source ./gtmenv B V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'set ^B(64)=1,^B(65)=1'
```

The shutdown sequence is as follows:

```
source ./gtmenv B V6.3-000A_x86_64
./originating_stop
source ./gtmenv A V6.3-000A_x86_64
```

```
./replicating_stop
source ./gtmenv P V6.3-000A_x86_64
./replicating_stop
```

Rollback automatically

A	B	P	Comments
O: ... *A95*, *A96*, *A97*, *A98*, *A99*	R: ... *A95*, *A96*, *A97*	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *A97*, *P37*, *P38*, *A98*, *P39*, *P40*	*A* as an originating primary instance at transaction number *A99*, replicates to *B* as a BC replicating secondary instance at transaction number *A97* and *P* as a SI that includes transaction number *A98*, interspersed with locally generated updates. Updates are recorded in each instance"s journal files using before-image journaling.
R: Rolls back to *A97* with *A98* and *A99* in the Unreplicated Transaction Log.	O: *A95*, *A96*, *A97*	S:Rolls back *A98*, *P38*, and *P40*	Instances receiving a replication stream from *A* can be configured to rollback automatically when *A* performs an online rollback by starting the Receiver Server with -autorollback. If *P*"s Receiver Server is so configured, it will roll *A98*, *P39* and *P40* into an Unreplicated Transaction Log. This scenario is straightforward. With the -noresync qualifier, the Receiver Server can be configured to simply resume replication without rolling back.

The following example run this scenario.

```
source ./gtmenv A V6.3-000A_x86_64
./db_create
./repl_setup
./originating_start A P 4000
./originating_start A B 4001
source ./gtmenv B V6.3-000A_x86_64
./db_create
./repl_setup
./replicating_start B 4001
source ./gtmenv A V6.3-000A_x86_64
./backup_repl startA
source ./gtmenv P V6.3-000A_x86_64
./db_create
./suppl_setup P startA 4000 -updok
$gtm_dist/mumps -r %XCMD 'for i=1:1:38 set ^P(i)=i'
source ./gtmenv A V6.3-000A_x86_64
$gtm_dist/mumps -r %XCMD 'for i=1:1:97 set ^A(i)=i'
source ./gtmenv B V6.3-000A_x86_64
./replicating_stop
source ./gtmenv A V6.3-000A_x86_64
$gtm_dist/mumps -r %XCMD 'set ^A(98)=50'
source ./gtmenv P V6.3-000A_x86_64
$gtm_dist/mumps -r %XCMD 'for i=39:1:40 set ^P(i)=i'
./replicating_stop
source ./gtmenv A V6.3-000A_x86_64
$gtm_dist/mumps -r %XCMD 'set ^A(99)=100'
./originating_stop
source ./gtmenv B V6.3-000A_x86_64
./originating_start B A 4001
./originating_start B P 4000
```

```
source ./gtmenv A V6.3-000A_x86_64
./replicating_start A 4001 -autorollback
source ./gtmenv P V6.3-000A_x86_64
#./rollback 4000 backward
./replicating_start_suppl_n P 4000 -updok -autorollback
#./replicating_start_suppl_n P 4000 -updok
```

The shutdown sequence is as follows:

```
source ./gtmenv A V6.3-000A_x86_64
./replicating_stop
source ./gtmenv P V6.3-000A_x86_64
./replicating_stop
source ./gtmenv B V6.3-000A_x86_64
./originating_stop
```

Switchover possibilities in a B←A→P→Q replication configuration

Consider a situation where A and P are located in one data center, with BC replication to B and Q respectively, located in another data center. When the first data center fails, the SI replication from A to P is replaced by SI replication from B to Q. The following scenario describes a switchover from B←A→P→Q to A←B→Q→P with unreplicated updates on A and P.

A	B	P	Q	Comments
O: ... *A95*, *A96*, *A97*, *A98*, *A99*	R: ... *A95*, *A96*, *A97*, *A98*	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *P37*, *A97*, *P38*	R: ... *P34*, *A95*, *P35*, *P36*, *A96*, *P37*	*A* as an originating primary instance at transaction number *A99*, replicates to *B* as a BC replicating secondary instance at transaction number *A98* and *P* as a SI that includes transaction number *A97*, interspersed with locally generated updates. *P* in turn replicates to *Q*.
Goes down with the data center	O: ... *A95*, *A96*, *A97*, *A98*, *B61*, *B62*	Goes down with the data center	... *P34*, *A95*, *P35*, *P36*, *A96*, *P37*	When a data center outage disables *A*, and *P*, *B* becomes the new originating primary, with *A98* as the latest transaction in its database and starts processing application logic to maintain business continuity. *Q* can receive the SI replication stream from *B*, without requiring a rollback since the receiver is not ahead of the source.
-	O: ... *A95*, *A96*, *A97*, *A98*, *B61*, *B62*	-	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *P37*, *A97*, *A98*, *Q73*, *B61*, *Q74*, *B62*	*Q* receives SI replication from *B* and also applies its own locally generated updates. Although *A97* and *A98* were originally generated on *A*, *Q* receives them from *B*. *Q* also computes and applies locally generated updates
... *A95*, *A96*, *A97*, *A98*, *A99*	O: ... *A95*, *A96*, *A97*, *A98*, *B61*, *B62*, *B63*, *B64*	... *P34*, *A95*, *P35*, *P36*, *A96*, *P37*, *A97*,*A98*, *P38*	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *P37*, *A97*, *A98*, *Q73*, *B61*, *Q74*, *B62*, *Q75*, *B63*, *Q76*, *B64*	While *B* and *Q*, keep the enterprise in operation, the first data center is recovered. Since *A* has transactions in its database that were not replicated to *B* when the latter started operating as the originating primary instance, and since *P* had transactions that were not replicated

A	B	P	Q	Comments
				to *Q* when the latter took over, *A* and *P* must now rollback their databases and create Unreplicated Transaction Files before receiving BC replication streams from *B* and *Q* respectively. *A* rolls off *A99*, *P* rolls off *P38*.
R: ... *A95*, *A96*, *A97*, *B61*, *B62*, *B63*, *B64*	O: ... *A95*, *A96*, *A97*, *B61*, *B62*, *B63*, *B64*, B65	R: ... *P34*, *A95*, *P35*, *P36*, *A96*, *P37*, *A97*, *A98*, *Q73*, *B61*, *Q74*, *B62*, *Q75*, *B63*, *Q76*, *B64*	S: ... *P34*, *A95*, *P35*, *P36*, *A96*, *P37*, *A97*, *A98*, *Q73*, *B61*, *Q74*, *B62*, *Q75*, *B63*, *Q76*, *B64*, *Q77*	Having rolled off their transactions into Unreplicated Transaction Logs, *A* can now operate as a BC replicating instance to *B* and *P* can operate as the SI replicating instance to *Q*. *B* and *Q* continue operating as originating primary instance and supplementary instance. *P* automatically receives M38 after applying the Unreplicated Transaction Log (from *P*) to *Q*. *A* and *P* automatically receive A99 after applying the Unreplicated Transaction Log (from *A*) to *B*.

The following example runs this scenario.

```
source ./gtmenv A V6.3-000A_x86_64
./db_create
./repl_setup
./originating_start A P 4000
./originating_start A B 4001
source ./gtmenv B V6.3-000A_x86_64
./db_create
./repl_setup
./replicating_start B 4001
source ./gtmenv A V6.3-000A_x86_64
./backup_repl startA
source ./gtmenv P V6.3-000A_x86_64
./db_create
./suppl_setup P startA 4000 -updok
./backup_repl startP
./originating_start P Q 4005
source ./gtmenv Q V6.3-000A_x86_64
./db_create
./suppl_setup Q startP 4005 -updnotok
source ./gtmenv A V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'for i=1:1:96 set ^A(i)=i'
source ./gtmenv P V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'for i=1:1:37 set ^P(i)=i'
source ./gtmenv Q V6.3-000A_x86_64
./replicating_stop
source ./gtmenv P V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'set ^P(38)=1000'
./replicating_stop
source ./gtmenv A V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'set ^A(97)=1000,^A(98)=1000'
source ./gtmenv B V6.3-000A_x86_64
./replicating_stop
source ./gtmenv A V6.3-000A_x86_64
```

```
$gtm_dist/mumps -r ^%XCMD 'set ^A(99)=1000'
./originating_stop
source ./gtmenv B V6.3-000A_x86_64
backup_repl startB
./originating_start B Q 4008
$gtm_dist/mumps -r ^%XCMD 'for i=1:1:62 set ^B(i)=i'
source ./gtmenv Q V6.3-000A_x86_64
./rollback 4008 backward
./suppl_setup Q startB 4008 -updok
$gtm_dist/mumps -r ^%XCMD 'for i=1:1:74 set ^Q(i)=i'
source ./gtmenv B V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'for i=63:1:64 set ^B(i)=i'
./originating_start B A 4004
source ./gtmenv A V6.3-000A_x86_64
./rollback 4004 backward
./replicating_start A 4004
source ./gtmenv Q V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'for i=75:1:76 set ^Q(i)=i'
./originating_start Q P 4007
./backup_repl startQ
source ./gtmenv P V6.3-000A_x86_64
./rollback 4007 backward
./replicating_start_suppl_n P 4007 -updnotok
source ./gtmenv Q V6.3-000A_x86_64
$gtm_dist/mumps -r ^%XCMD 'set ^Q(77)=1000'
cat A/gtm.lost
cat P/gtm.lost
```

The shutdown sequence is as follows:

```
source ./gtmenv P V6.3-000A_x86_64
./replicating_stop
source ./gtmenv A V6.3-000A_x86_64
./replicating_stop
source ./gtmenv Q V6.3-000A_x86_64
./replicating_stop
./originating_stop
source ./gtmenv B V6.3-000A_x86_64
./originating_stop
```

Changing the global directory in an A→B replication configuration

In a replication configuration, a global directory provides information to map global updates to their respective database files. As replication processes pick the state of the global directory at process startup, any change made to the global directory requires process restarts (at a minimum) to bring that change into effect. A switchover mechanism can ensure application availability while making global directory changes.

On B:

1. Shut down the Receiver Server and the Update Process.

2. Make a copy of the current global directory.

3. If the globals you are moving have triggers, make a copy of their definitions with MUPIP TRIGGER -SELECT and delete them with MUPIP TRIGGER.

4. Update the global directory.

5. If you are rearranging the global name spaces which do not contain any data, skip to step 9.

6. Create a backup copy of B, turn off replication, and cut the previous links of the journal file.

7. Use the MERGE command to copy a global from the prior to the new location. Use extended references (to the prior global directory) to refer to global in the prior location.

8. If the globals you are moving have triggers, apply the definitions saved in step 3.

9. Turn replication on for the region of the new global location.

10. Make B the new originating instance. For more information, refer to "Switchover possibilities in an A→B replication configuration" (page 268).

On A:

1. Shutdown replication.

2. If the globals you are moving have triggers, make a copy of their definitions with MUPIP TRIGGER -SELECT and delete them with MUPIP TRIGGER; note if the triggers are the same as those on B, which they normally would be for a BC instance you can just delete them and use the definitions saved on B.

3. Update the global directory.

4. If you are rearranging the global name spaces which do not contain any data, skip to step 7.

5. Create a backup copy of A, turn off replication, and cut the previous links of the journal file.

6. Use the MERGE command to copy a global from the prior to the new location. Use extended references (to the prior global directory) to refer to global in the prior location.

7. If the globals you are moving have triggers, apply the definitions saved in step 1.

8. Turn replication on for the region of the new global location.

9. Make A the new replicating instance.

Perform a switchover to return to the A->B configuration. Once normal operation resumes, remove the global from the prior location (using extended references) to release space.

If a switchover mechanism is not in place and a downtime during the global directory update is acceptable, follow these steps:

On B:

- Perform steps 1 to 9.

- Restart the Receiver Server and the Update Process.

On A:

- Bring down the application (or prevent new updates from getting started).

- Perform Steps 1 to 8.

- Restart the originating instance.

- Restart the active Source Server.

- Bring up the application.

This example adds the mapping for global ^A to a new database file A.dat in an A->B replication configuration.

```
source ./gtmenv A V6.3-000A_x86_64
./db_create
./repl_setup
./originating_start A B 4001
source ./gtmenv B V6.3-000A_x86_64
./db_create
./repl_setup
./replicating_start B 4001
source ./gtmenv A V6.3-000A_x86_64
$gtm_dist/mumps -r %XCMD 'for i=1:1:10 set ^A(i)=i'
./repl_status
source ./gtmenv B V6.3-000A_x86_64
./replicating_stop
cp B/gtm.gld B/prior.gld
$gtm_dist/mumps -r ^GDE @updgld
./db_create
mkdir backup_B
$gtm_dist/mupip backup "*" backup_B  -replinst=backup_B/gtm.repl
$gtm_dist/mupip set -journal=on,before_images,filename=B/gtm.mjl -noprevjnlfile -region "DEFAULT"
$gtm_dist/mumps -r %XCMD 'merge ^A=^|"B/prior.gld"|A'
$gtm_dist/mupip set -replication=on -region AREG
./originating_start B A 4001
source ./gtmenv A V6.3-000A_x86_64
./originating_stop
./rollback 4001 backward
cat A/gtm.lost  #apply lost transaction file on A.
./replicating_start A 4001
./replicating_stop
cp A/gtm.gld A/prior.gld
$gtm_dist/mumps -r ^GDE @updgld
./db_create
mkdir backup_A
$gtm_dist/mupip backup "*" backup_A -replinst=backup_A/gtm.repl
$gtm_dist/mupip set -journal=on,before_images,filename=A/gtm.mjl -noprevjnlfile -region "DEFAULT"
$gtm_dist/mumps -r %XCMD 'merge ^A=^|"A/prior.gld"|A'
$gtm_dist/mupip set -replication=on -region AREG
./replicating_start A 4001
./repl_status
#Perform a switchover to return to the A->B configuration. Remove the global in the prior location to release
 space
▶ with a command like Kill ^A=^|"A/prior.gld"|A'.
```

🔻

The shutdown sequence is as follows:

```
source ./gtmenv A V6.3-000A_x86_64
./replicating_stop
source ./gtmenv B V6.3-000A_x86_64
./originating_stop
```

Rolling Software Upgrade

A rolling software upgrade is the procedure of upgrading an instance in a way that there is minimal impact on the application uptime. An upgrade may consist of changing the underlying database schema, region(s), global directory, database version, application version, triggers, and so on. There are two approaches for a rolling upgrade. The first approach is to upgrade the replicating instance and then upgrade the originating instance. The second approach is to upgrade the originating instance first while its replicating (standby) instance acts as an originating instance.

The following two procedures demonstrate these rolling software upgrade approaches for upgrading an A→B replication configuration running an application using GT.M V6.1-000_x86_64 to GT.M V6.2-001_x86_64 with minimal (a few seconds) impact on the application downtime.

Upgrade the replicating instance first (A→B)

On B:

1. Shut down the passive Source and Receiver Servers and the application.

2. Turn off replication.

3. Perform a MUPIP RUNDOWN operation and make a backup.

4. Open DSE, run DUMP -FILEHEADER for each region (FIND -REGION=<Region_Name>) and note down the largest Region Seqno.

5. Upgrade the instance. An upgrade may include include adding triggers, adding/removing regions, changing GDE mapping, and so on.

6. Open DSE again, run DSE DUMP -FILEHEADER for each region (FIND -REGION=<Region_Name>) and note down the largest Region Seqno. If the largest Region Seqno noted in step 4 and largest Region Seqno noted in this step are the same, proceed to step 7. Otherwise, execute DSE CHANGE -FILEHEADER - REG_SEQNO=<Largest_Region_Seqno_from_step_4> for the region having the largest Region Seqno.

7. Cut the back links to the prior generation journal files with a command like:

    ```
    $gtm_dist/mupip set -journal=on,before_images,filename=B/gtm.mjl -noprevjnlfile -region "DEFAULT"
    ```

8. Turn on replication.

9. If the use of replication filters apply to your situation, bring up the replicating instance with the new-to-old filter on the Source Server of A, and the old-to-new filter on the Receiver Server of B. Otherwise, bring up the replicating instance on B.

10. Wait for B to automatically catup up the pending updates from A.

on A:

1. When there are no/low updates on A, shut down the Source Server.

2. Turn off replication.

3. Perform a MUPIP RUNDOWN and make a backup copy of the database.

4. Perform a switchover to make B the originating instance. Apply lost/broken transactions, if any, on B.

5. Open DSE, run DUMP -FILEHEADER for each region (FIND -REGION=<Region_Name>) and note down the largest Region Seqno.

6. Upgrade the instance. An upgrade may include include adding triggers, adding/removing regions, changing GDE mapping, and so on.

7. Open DSE again, run DSE DUMP -FILEHEADER for each region (FIND -REGION=<Region_Name>) and note down the largest Region Seqno. If the largest Region Seqno noted in step 5 and largest Region Seqno noted in this step are the same, proceed to step 8. Otherwise, execute DSE CHANGE -FILEHEADER -REG_SEQNO=<Largest_Region_Seqno_from_step_5> for the region having the largest Region Seqno.

8. Cut the back links to the prior generation journal files with a command like:

```
$gtm_dist/mupip set -journal=on,before_images,filename=A/gtm.mjl -noprevjnlfile -region DEFAULT
```

9. Turn on replication.

10. Start the Receiver Server of A.

Upgrade the originating instance first (A→B)

on A:

1. When there are no updates on A and both A and B are in sync, shut down the Source Server.

2. Turn off replication.

3. Perform a MUPIP RUNDOWN and make a backup copy of the database.

4. Perform a switchover to make B the originating instance. This ensure application availability during the upgrade of A.

5. Open DSE, run DUMP -FILEHEADER for each region (FIND -REGION=<Region_Name>) and note down the largest Region Seqno.

6. Upgrade the instance. An upgrade may include include adding triggers, adding/removing regions, changing GDE mapping, upgrading the GT.M version, and so on.

7. Open DSE again, run DSE DUMP -FILEHEADER for each region (FIND -REGION=<Region_Name>) and note down the largest Region Seqno. If the largest Region Seqno noted in step 5 and largest Region Seqno noted in this step are the same, proceed to step 8. Otherwise, execute DSE CHANGE -FILEHEADER -REG_SEQNO=<Largest_Region_Seqno_from_step_5> for the region having the largest Region Seqno.

8. Cut the back links to the prior generation journal files with a command like:

```
$gtm_dist/mupip set -journal=on,before_images,filename=A/gtm.mjl -noprevjnlfile -region DEFAULT
```

9. Turn on replication.

10. If the use of replication filters apply to your situation, bring up the Receiver Server with the old-to-new filter. Otherwise bring up the Receiver Server.

11. Wait for A to automatically catch up the pending updates from B.

on B:

1. When there are no/low updates on A, shut down the Source Server.

2. Turn off replication.

3. Perform a MUPIP RUNDOWN and make a backup copy of the database.

4. Perform a switchover to reinstate A as the originating instance. This ensures application availability during the upgrade of B.

5. Open DSE, run DUMP -FILEHEADER for each region (FIND -REGION=<Region_Name>) and note down the largest Region Seqno.

6. Upgrade the instance. An upgrade may include include adding triggers, adding/removing regions, changing GDE mapping, and so on.

7. Open DSE again, run DSE DUMP -FILEHEADER for each region (FIND -REGION=<Region_Name>) and note down the largest Region Seqno. If the largest Region Seqno noted in step 5 and largest Region Seqno noted in this step are the same, proceed to step 8. Otherwise, execute DSE CHANGE -FILEHEADER -REG_SEQNO=<Largest_Region_Seqno_from_step_5> for the region having the largest Region Seqno.

8. Cut the back links to the prior generation journal files with a command like:

    ```
    $gtm_dist/mupip set -journal=on,before_images,filename=B/gtm.mjl -noprevjnlfile -region DEFAULT
    ```

9. Turn on replication.

10. Start the Receiver Server of B.

11. The upgrade of A and B is complete.

⚠️ Note on Triggers

> While adding triggers, bear in mind that triggers get replicated if you add them when replication is turned on. However, when you add triggers when replication is turned off, those triggers and the database updates resulting from the executing their trigger code do not get replicated.

Here is an example to upgrade A and B deployed in an A→B replication configuration from V6.1-000_x86_64 to V6.2-001_x86_84. This example uses instructions from the "Upgrade the originating instance first (A→B)" (page 282) procedure.

```
source ./env A V6.1-000_x86_64
./db_create
./repl_setup
./originating_start A B 4001
source ./env B V6.1-000_x86_64
./db_create
./repl_setup
```

```
./replicating_start B 4001
source ./env A V6.1-000_x86_64
$gtm_dist/mumps -r %XCMD 'for i=1:1:100 set ^A(i)=i'
./status
source ./env B V6.1-000_x86_64
./replicating_stop
source ./env A V6.1-000_x86_64
./status
./originating_stop
$gtm_dist/mupip set -replication=off -region "DEFAULT"
$gtm_dist/dse dump -f 2>&1| grep "Region Seqno"
#Perform a switchover to make B the originating instance.
source ./env A V6.2-001_x86_64
$gtm_dist/mumps -r ^GDE exit
$gtm_dist/mupip set -journal=on,before_images,filename=A/gtm.mjl -noprevjnlfile -region "DEFAULT"
#Perform the upgrade
$gtm_dist/dse dump -fileheader 2>&1| grep "Region Seqno"
#If Region Seqno is greater than the Region Seqno noted previously, run $gtm_dist/dse change -fileheader
▶ -req_seqno=<previously_noted_region_seqno>.
./repl_setup
#A is now upgraded to V6.2-001_x86_64 and is ready to resume the role of the originating instance. Shutdown B and
▶ reinstate A as the originating instance.
./originating_start A B 4001
source ./env B V6.2-001_x86_64
$gtm_dist/mumps -r ^GDE exit
$gtm_dist/mupip set -journal=on,before_images,filename=B/gtm.mjl -noprevjnlfile -region "DEFAULT"
#Perform the upgrade
$gtm_dist/dse dump -fileheader 2>&1| grep "Region Seqno"
#If Region Seqno is different, run $gtm_dist/dse change -fileheader -req_seqno=<previously_noted_region_seqno>.
$gtm_dist/dse dump -f 2>&1| grep "Region Seqno"
#If Region Seqno is greater than the Region Seqno noted previously, run $gtm_dist/dse change -fileheader
▶ -req_seqno=<previously_noted_region_seqno>.
./repl_setup
./replicating_start B 4001
```

The shutdown sequence is as follows:

```
source ./env B V6.2-001_x86_64
./replicating_stop
source ./env A V6.2-001_x86_64
./originating_stop
```

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).

- Shutdown the originating instance servers (all active and/or passive Source Servers).

- On its replicating instances, ensure that there are no GT.M or MUPIP 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.

Creating a new Replication Instance File

You do not need to create a new replication instance file except when you upgrade from a GT.M version prior to V5.5-000. Unless stated in the release notes of your GT.M version, you instance file does not need to be upgraded. If you are creating a new replication instance file for any administration purpose, remember that doing do so will remove history records which may prevent it from resuming replication with other instances. To create a new replication instance file, follow these 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.

- 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 described in other examples of this section):

  - If this is instance is to receive SI replication or to receive BC replication from an instance that receives SI replication, use the command:

    ```
    mupip replicate -instance_create -supplementary
    ```

  - Otherwise use the command:

    ```
    mupip replicate -instance_create
    ```

  - If a replication instance file already exists, these commands will create a backup copy of the replicating instance and then create a new replication instance file. If you want to prevent accidental overwriting your existing replication instance file, use the -noreplace qualifier with these commands.

- Prepare it to accept a replication stream:

  - Start a passive Source Server. If this is an SI replicating instance, use the -updok flag to start the passive Source Server.

  - Start the Receiver Server using the updateresync. For versions prior to V5.5-000 use the -updateresync qualifier and for GT.M versions V5.5-000 or newer use -updateresync=<repl_inst>.For example, mupip replicate -receiver -start -updateresync=<repl_inst> where repl_inst is the prior replication file if the source is V5.5-000 or newer and -updateresync if it is an older GT.M release.

- 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.

On instances with the same endian-ness, follow these steps to create a replication instance file without using the -updateresync qualifier.

On the source side:

- Use the MUPIP BACKUP command with the -REPLINSTANCE qualifier to backup the instance to be copied. The source server for the instance must be started at least once before backing up the replication instance file.

- 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.

> When the instances have different endian-ness, create a new replication instance file as described in "Creating the Replication Instance File" (page 294)

Setting up a secured TLS replication connection

The following example creates two instances (Alice and Bob) and a basic framework required for setting up a TLS replication connection between them. Alice and Bob are fictional characters from https://en.wikipedia.org/wiki/Alice_and_Bob and represent two instances who use the certificates signed by the same demo root CA. This example is 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. 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.

1. Remove the comment tags from the following lines in the gtmenv script:

```
export gtmcrypt_config=$PWD/$gtm_repl_instname/config_file
echo -n "Enter Password for gtmtls_passwd_${gtm_repl_instname}: ";export
▶ gtmtls_passwd_${gtm_repl_instname}="`$gtm_dist/plugin/gtmcrypt/maskpass|tail -n 1|cut -f 3 -d " "`"
```

▼

2. Execute the gtmenv script as follows:

```
$ source ./gtmenv Alice V6.2-001_x86_64
```

This creates a GT.M environment for replication instance name Alice. When prompted, enter a password for gtmtls_passwd_Alice.

3. `./db_create`

   This creates the global directory and the database for instance Alice.

4. Create a demo root CA, leaf-level certificate, and a $gtmcrypt_config file with a tlsid called Alice for instance Alice. Note that in this example, $gtmcrypt_config is set to $PWD/Alice/config_file. For more information on creating the $gtmcrypt_config file and the demo certificates required to run this example, refer to Appendix G: "Creating a $gtmcrypt_config file" (page 514).

   Your $gtmcrypt_config file should look something like:

```
tls: {
    verify-depth: 7;
     CAfile: "/path/to/certs/ca.crt";
     Alice: {
            format: "PEM";
            cert: "/path/to/certs/Alice.crt";
            key: "/path/to/certs/Alice.key";
     };
  };
```

5. Turn replication on and create the replication instance file:

   `$ ./repl_setup`

6. Start the originating instance Alice:

   `$ ./originating_start Alice Bob 4001 -tlsid=Alice -reneg=2`

On instance Bob:

1. Execute the gtmenv script as follows:

   `$ source ./gtmenv Bob V6.2-001_x86_64`

   This creates a GT.M environment for replication instance name Bob. When prompted, enter a password for gtmtls_passwd_Bob.

2. `$ ./db_create`

   This creates the global directory and the database for instance Bob.

3. Create a leaf-level certificate and a $gtmcrypt_config file with a tlsid called Bob for instance Bob. Note that in this example, $gtmcrypt_config is set to $PWD/Bob/config_file. Note that you would use the demo CA that you created before to sign this leaf-level certificate. For replication to proceed, both leaf-level certificates must be signed by the same root CA. For more information, refer to Appendix G: "Creating a $gtmcrypt_config file" (page 514).

   Your $gtmcrypt_config file should look something like:

```
tls: {
    verify-depth: 7;
     CAfile: "/path/to/certs/ca.crt";
     Bob: {
            format: "PEM";
            cert: "/path/to/certs/Bob.crt";
            key: "/path/to/certs/Bob.key";
```

```
        };
    };
```

4. Turn replication on and create the replication instance file:

```
$ ./repl_setup
```

5. Start the replicating instance Bob.

```
$ ./replicating_start Bob 4001 -tlsid=Bob
```

For subsequent environment setup, use the following commands:

```
source ./gtmenv Bob V6.2-001_x86_64 or source ./gtmenv Alice V6.2-001_x86_64
./replicating_start Bob 4001 -tlsid=Bob or ./originating_start Alice Bob 4001 -tlsid=Alice -reneg=2
```

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 249).

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.

To make storage space available, first consider moving unwanted non-journaled 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.

Rollback data from crashed (idle) regions

When a rollback operations fails with CHNGTPRSLVTM, NOPREVLINK, and JNLFILEOPENERR messages, evaluate whether you have a crashed region in your global directory that is seldom used for making updates (idle). The updates in an idle region's current generation journal file may have timestamps and sequence numbers that no longer exist in the prior generation journal file chains of more frequently updated regions because of periodic pruning of existing journal files as part of routine maintenance. MUPIP SET and BACKUP commands can also remove previous generation journal file links.

Terminating a process accessing an idle region abnormally (say with kill -9 or some other catastrophic event) may leave its journal files improperly closed. In such an case, the discrepancy may go unnoticed until the next database update or rollback. Performing a rollback including such an idle region may then resolve the unified rollback starting time, (reported with a CHNGTPRSLVTM message), to a point in time that does not exist in the journal file chain for the other regions, thus causing the rollback to fail.

In this rare but possible condition, first perform a rollback selectively for the idle region(s). Here are the steps:

1. Create a temporary global directory which maps only the idle region(s) by mapping one, often the only, such idle region's file to the default region.

2. Set the environment variable gtmgbldir to point to the temporary global directory.

3. Perform an optimal rollback (MUPIP JOURNAL -ROLLBACK -BACKWARD "*")

4. Analyze and process any broken or lost transaction files that the rollback procedure may generate.

5. Set the environment variable gtmgbldir to point back to the location of the global directory for your application.

6. Perform a normal rollback for your application.

You do not need to perform these steps if you have a non-replicated but journaled database because RECOVER operations do not coordinate across regions.

As a general practice, perform an optimal recovery/rollback every time when starting a GT.M application from quiescence and, depending on the circumstances, after a GT.M process terminates abnormally.

FIS recommends rotating journal files with MUPIP SET when removing old journal files or ensuring that all regions are periodically updated.

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 use 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.

- 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.

- 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.

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" [88].

   -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 288).

Example:

```
$ mupip set -replication=on -file mumps.dat
```

This example enables database replication and turns before-image journaling on for mumps.dat.

```
$ 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] [-noqdbrundown]
```

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 either gtm_repl_instname is not set or -name is not specified, GT.M produces an error. For more information, refer to the entries for gtm_repl_instname and gtm_repl_instance in "Environment Variables" (page 19).

-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.

-noqdbrundown

Permits more than 32,767 udpating processes to concurrently the replication instance file and database file(s). The default (-noqdbrundown) permits up to 32,767 concurrent updating processes to access a database file or a replication instance file.

This permission is effected by the QDBRUNDOWN flags in database file headers and in replication instance files. When an open database file or replication instance file with QDBRUNDOWN set is first concurrently accessed by more than 32,767 processes, GT.M:

1. logs a NOMORESEMCNT message in the system log, and

2. stops counting the number of attached processes. This means that GT.M cannot recognize when the number of attached processes reaches zero (0) in order to release the corresponding shared resources, and therefore requires explicit manual clean up of resources for an orderly shutdown.

Except in application configurations that require it, FIS recommends not setting QDBRUNDOWN. Not setting QDBRUNDOWN allows GT.M to clean up resources, instead of putting the burden on the operational procedures. Where GT.M cannot perform an orderly shutdown, an explicit, clean shutdown must be performed as follows:

1. Replicated instances require a mupip journal -rollback -backward "*" executed after the mupip replicate -source -shutdown command (remember that even instances receiving a replication stream have one or more Source Servers).

2. Database files that are journaled but not part of a replication instance require a mupip journal -recover -backward command.

3. For database files that are not journaled (and hence not replicated), perform a mupip rundown, or if appropriate, delete and recreate them.

Example:

```
$ export gtm_repl_instance=mutisite.repl
$ export gtm_repl_instname=America
$ mupip replicate -instance_create
```

This example creates a replication instance file called multisite.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=]|-cleanslots}
 [-name=<new-name>]
 [-[no]qdbrundown]
```

   -editinstance

Displays or changes the attributes of the specified *instance-file*. Use -editinstance in combination with the SHOW, CHANGE, or CLEANSLOTS qualifiers.

   -cleanslots

Goes through all 16 slots in the replication instance file, identifies the slots that are inactive, and clears them to make them available for reuse.

> ### Note
>
> Initially, all the slots are unused. A Source Server replicating to a secondary instance for the first time utilizes an unused slot to store the information related to that secondary. Any Source Server process replicating to the same secondary instance updates information using the same slot until -CLEANSLOT clears the slot to

make it reusable. Use -CLEANSLOT to clear inactive slots when the originating instance connects to more than 16 different secondary instances throughout its lifetime.

-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.

-[no]qdbrundown

Controls the setting of the "HDR Quick database rundown is active" field in a replication instance file. For more information about this setting, refer to "Creating the Replication Instance File" [294].

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

The MUPIP REPLICATE -SOURCE -START command starts a Source Server process. The general syntax for starting a Source Server in active mode is:

```
mupip replicate -source -start
[connection_qualifiers]
[instance_freeze_qualifiers]
[jnlpool_setup_qualifiers]
log_file_management_qualifiers
[primary_role_qualifiers]
secondary_identification_qualifiers
[tls_replication_qualifiers]
[replication_filter_qualifier]
```

The square brackets [] denote an optional qualifier group. The optional and mandatory qualifiers in each qualifier group are as follows:

Category	Qualifiers
connection_qualifiers	• **[-cmplvl=<compression_level>]** • **[-connectparams=<hard_tries>, <hard_tries_period>, <soft_tries_period>, <alert_time>, <heartbeat_period>, <max_heartbeat_wait>]**
jnlpool_setup_qualifiers	• **[-buffsize=<Journal Pool size in bytes>]** • **[-[no]jnlfileonly]**
logfile_management_qualifiers	• **-log=<log file name>** • **[-log_interval=<integer>]**
primary_role_qualifiers	• **{-rootprimary|-propagateprimary}** • **[{-updok|-updnotok}]**

Category	Qualifiers
replication_filter_qualifier	• **-filter=<filter command>**
secondary_identification_qualifiers	• **-instsecondary=<secondary_instance_name>** • **-secondary=<hostname:port>**
tls_replication_qualifiers	• **[-[no]plaintextfallback]** • **[-renegotiate_interval=<minutes>]** • **-tlsid=<label>**

You can start the Source Server in passive mode with mupip replicate -source -start -passive. The general syntax for starting a Source Server in passive mode is the same as that of starting a Source Server in active mode except that -passive is not compatible with -secondary=<hostname:port>.

Connection Qualifiers:

    -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 compression 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 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.

By default, GT.M searches for the libz.so shared library in the standard system library directories (for example, /usr/lib, /usr/local/lib, /usr/local/lib64). If the shared library is installed in a non-standard location, before starting replication, you must ensure that the environment variable $LIBPATH (AIX and z/OS) or $LD_LIBRARY_PATH (other UNIX platforms) includes the directory containung 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.

-connectparams=<hard tries>,<hard tries period>,<soft tries period>,<REPLALERT time>,<heartbeat period>,<max heartbeat wait>

Specifies the connection parameters as a comma delimited series of integer values. If the connection between the Source and Receiver Servers breaks or the Source Server fails to connect to the Receiver Server at startup, the Source uses the first four parameters for reconnection attempts as follows:

• It makes the number of new connection attempts specified by the <hard tries> value every <hard tries period> milliseconds.

• Then, the Source Server attempts connection every <soft tries period> seconds and logs the REPLALERT message every <alert time> seconds.

• The only difference between hard and soft connection attempts is their interval duration measurement units (milliseconds vs. seconds) and the association of the REPLALERT message with failed soft connection attempts.

• The presence of the REPLALERT message in the Source Server log file can be used with a monitoring tool to alert the operator about an unaccessible Receiver Server.

• Specifying 0 for <alert time> disables the REPLALERT messages. By default, REPLAERT messages are disabled. Specifying 0 for <hard tries> disables making hard connection attempts which is useful in a situation where the early of logging of the REPLALERT messages is relevant.

The Source Server uses the fifth and sixth parameters when the connection is established with the Receiver Server. It send 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. If the response does not arrive within <max heartbeat wait>, the Source Server closes the connection and starts to establish a new conection using the first four parameters. The Source Server temporarily stalls the heartbeat message interchange mechanism during events such as secured renegotiation or online rollback. The stalling of heartbeat connection attempts may also occur when the Receiver Server is not started or is not accessible on the network.

GT.M automatically adjusts the values for <soft tries period> and <hard tries period> to about half of the GT.M determined shutdown wait time. GT.M logs these adjustments in the Source Server log file. This helps prevent inadvertent timeouts of a Source Server shutdown due to high <soft tries period> and/or <hard tries period> and prevents out-of-design conditions.

The default -CONNECTPARAMS values are 5,500,5,0,15,60. With this setting, the Source Server makes 5 hard connection attempts in 500 milliseconds intervals, and then makes soft connection attempts every 5 seconds without logging the REPLALERT message. Once the connection is established, the Source Server sends a heartbeat messages every 15 seconds and closes the connection if a response does not arrive in 60 seconds. Then, the Source Server starts establishings a new connection with hard connection attempts.

You can specify up to six parameters with -CONNECTPARAMS. MUPIP uses the defaults for the parameters not specified.

Journal Pool Setup Qualifiers:

  -buffsize=<Journal Pool size in bytes>

Specifies the size of the Journal Pool. The server rounds the size up or down to suit to align with other specified or implicit requirements. Any size less than 1 MiB is rounded up to 1 MiB. If -BUFFSIZE is not specified, MUPIP sets the Journal Pool size to 64 MiB. 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. The maximum value is 68719476736 (64GiB). MUPIP REPLICATE -SOURCE ignores -BUFFSIZE if the Journal Pool is already set up. The first Source Server process started on an instance sets up the Journal Pool.

-[no]jnlfileonly

Forces the Source Server to read transactions from journal files instead of journal pool shared memory. When combined with the SYNC_IO journal option, this feature delays replication of transactions until their journal records are hardened to disk. This may be useful when replicating to a supplementary instance, as a crash and rollback on the primary could otherwise necessitate a rollback of local updates on the receiving instance. The default is -NOJNLFILEONLY.

Log File Management Qualifiers:

-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. The Source Server also directs errors to the log file. Operators should check the Source Server log file for errors.

-log_interval=<integer>

Specifies the number of transactions for which the Source Server should leave between writing progress to the log file. The default logging interval is 1000 transactions.

-log_interval=0 sets the logging interval to the default value.

Replication Filter Qualifiers:

-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.

You can also define an environment variable like **export gtmfilter=""$gtm_dist/mumps -run ^OLD2NEW^FILTER""** to bypass shell escaping and use it in the MUPIP command as follows:

```
$gtm_dist/mupip replic -receiver -start -listenport=$portno -filter=$gtmfilter -log=$REC_LOG_FILE
```

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 the following deploys this filter on the Receiver Server:
  ; export gtmfilter=""$gtm_dist/mumps -run ^extfilter""
  ; $gtm_dist/mupip replic -receiver -start -buffsize=$gtm_buffsize
  ;-listenport=$portno
  ;-filter=$gtmfilter -log=$REC_LOG_FILE
  ; deploys this demo filter on the Receiver Server.
  ;
  ; This program is a simple filter program that receives journal records
  ; from standard input and write them back to standard output.
  ; Journal records between TSTART
  ; and TCOMMIT are written with single write command.
  ;
  ; Do not use this filter program as-is in a production environment.
  ; While writing filter programs ensure that your programming logic never
  ; exposes non-public information (NPI) stored in GT.M local variables
  ; and global variables. You should take additional care about
  ; managing and sharing GTM_FATAL_ERROR.ZSHOW_DMP_*.txt
  ; and core dump files that may contain NPI.
  new TSTART,TCOMMIT,EOT,EOL,log,extRec,filtrOut,cntr,tmp,rectype,max
  set $ztrap="goto err"
  set TSTART="08"
  set TCOMMIT="09"
  set EOT="99"
  set max=1024*1024                 ; maximum size of a local variable;
  set log=$ztrnlnm("filterlog")   ; use environment variable "filterlog" (if defined) to indicate which logfile to
 use
  set:log="" log="/dev/null"        ; if filterlog is not defined, write log file output to /dev/null
  set EOL=$char(10)
  open log:newversion
  use $principal:(nowrap:width=max)
  for  do
  . use $principal
  . read extRec
  . if $zeof halt
  . set rectype=$piece(extRec,"",1)
  . if rectype'=EOT do
  .. if rectype'=TSTART set filtrOut=extRec_EOL
  .. else  do  ; Write Journal Records within TSTART and TCOMMIT boundary in one go (1MiB limit).
  ... set filtrOut=extRec_EOL
  ... for  read extRec set filtrOut=filtrOut_extRec_EOL quit:$zextract(extRec,1,2)=TCOMMIT
  ... if $zeof halt
  .. ; set $x=0 is needed so every write starts at beginning of record position
  .. ; don't write more than "width" characters in one output operation to avoid "chopping" of output
  .. ; and/or eol in the middle of output stream
  .. ; default width=1024*1024
  .. ; use $zsubstr to chop at valid character boundary (single or multi byte character)
  .. set tmp=filtrOut
  .. for  quit:tmp=""  do
  ... set $x=0,record($increment(cntr))=$zsubstr(tmp,1,max-1),tmp=$zextract(tmp,$zlength(record(cntr))+1,
$zlength(tmp))
  ... write record(cntr)
  ... kill record(cntr)
  . use log
  . write "Received: ",EOL,$select(rectype'=TSTART:extRec_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 "*"
 halt
```

This example reads logical database updates associated with a transaction from STDIN and writes them to a filter log (if filterlog environment variable is defined) 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.

Primary Role Qualifiers:

-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.
>
> - 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.

-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.

-updnotok

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.

-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.

Secondary Identification Qualifiers:

-instsecondary

Specifies the replicating instance to which the Source Server replicates data. -instsecondary also identifies a Source Server process target for actions like changing to active mode or passive mode, checking Source Server health, or shutting down a Source Server.

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 (say by a kill -9). Any Source Server that was kill -15ed or MUPIP 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. 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.

-secondary=<hostname:port>

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.

TLS Replication Qualifiers:

-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 gtmcrypt_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 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 approximate time in minutes to wait before attempting to perform a TLS renegotiation. The default -RENEGOTIATE_INTERVAL is a little over 120 minutes. The minimum -RENEGOTIATE_INTERVAL is 1 minute. A value of zero causes GT.M never to attempt a renegotiation. The MUPIP 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.

> ⚠️ TLSv1.3 introduces a number of protocol changes. The concept of renegotiation was removed. As a result, when using TLSv1.3, this qualifier takes on a different meaning. GT.M used TLSv1.2 renegotiation to ensure appropriate refresh of session keys (for rationale, see Luykx, A. and K. Paterson, "Limits on Authenticated Encryption Use in TLS", 2016; http://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf). As a result, GT.M uses the term "renegotiation" for SOCKET devices where the mode is "server" and "renegotiation_interval" for the

frequency of renegotiation during database replication. For TLSv1.3 sessions, the GT.M Reference TLS plugin treats the "renegotiation" as a request to update the session keys.

OpenSSL 3.0 by default does not allow client-side initiated TLSv1.2 renegotiation requests due to potential DoS attacks. Because of this, the GT.M Reference TLS plugin built with a pre-OpenSSL 3.0 version does not support client initiated renegotiation when communicating with a plugin build with OpenSSL 3.0. This limitation only affects database replication and not SOCKET devices. There is currently no work-around for this limitation.

Shutting down the Source Server

Command syntax:

```
mupip replicate -source -shutdown
 [-timeout=<timeout in seconds>]
 [-zerobacklog]
```

Qualifiers:

-shutdown

Shuts down the Source Server.

-timeout=<timeout in seconds>

Specifies the time (in seconds) the shutdown command should wait before signaling the Source Server to shut down. If you specify -timeout=0, MUPIP initiates the shutdown immediately. The default is 120 seconds and the maximum timeout is 3600 seconds. Any value higher than 3600 produces the INVSHUTDOWN error.

After initiating shutdown of the Source Server, GT.M waits up to 90 * number_of_regions seconds (with a minimum of 120 seconds) for the Source Server process to exit normally.

-zerobacklog

Restricts Journal Pool in a way that only existing processes can complete their updates and shuts down the Source Server:

- with the REPL0BACKLOG message as soon as there is no backlog and the Source Server has received acknowledgement of all its updates from the Receiver Server.

- with the REPLBACKLOG message when the timeout expires while there was a backlog and/or unacknowledged updates from the Receiver Server.

The REPL0BACKLOG message helps confirm that two instances are in sync (seqno-wise) because all Source Server updates have reached the Receive Pool at the time of shutdown and there are no-inflight updates. The REPL0BACKLOG message also causes the timeout to expire prematurely. In planned switchover scenarios, the REPL0BACKLOG message assists in operationally preventing lost transaction file generation.

Activating a Passive Source Server

Command syntax:

```
mupip replicate -source -activate
```

```
-secondary=<hostname:port>
-log=<log file name>
[-log_interval=<integer>]
-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.

-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.

-log_interval

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 default value.

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 -instsecondary 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.

Stopping the Receiver Filter

Command Syntax:

```
mupip replicate -receiver -stopreceiverfilter
```

Qualifiers:

-stopreceiverfilter

Turns off any active filter on the Receiver Server without turning off the Receiver Server. Using -stopreceiverfilter when no filter is active returns a non-success return code. -stopreceiverfilter is not compatible with any other -receiver qualifier.

Freezing an Instance

The MUPIP REPLICATE -SOURCE command with instance freeze qualifiers manage the Instance Freeze facility of an instance. The syntax is:

```
mupip replicate -source
  [-[no]comment[='"<string>"']]
  -freeze[=on|off]
```

Qualifiers:

-[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.

-freeze[=on|off]

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.

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 150).

For more information on Instance Freeze, refer to "Instance Freeze" (page 254).

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>]
```

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.

-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>
```

▼

Qualfiers:

-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 default 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 default value.

Reporting the Current Backlog of Journal Records

Command syntax:

```
mupip replicate -source -showbacklog
```

Qualifiers:

-showbacklog

Reports four fields that help determine the backlog of updates on the Source Server. These fields are:

1. Current backlog of journal records (in terms of the number of JNL_SEQNOs)

2. Last JNL_SEQNO written to the Journal Pool

3. Last JNL_SEQNO sent by the Source Server to the Receiver Server

4. Last JNL_SEQNO acknowledged by the Receiver Server

The backlog is the difference between the second and third fields. When backlog is 0 and the remaining three fields have the same value, it means that all Source Server updates have reached the Receiver Pool and there are no in-flight updates.

> ⚠️ **Important**
>
> A -SHOWBACKLOG operation momentarily restricts access to the Journal Pool to determine the backlog. If your operations demand frequent checking for replication backlogs, FIS recommends using the ^%PEEKBYNAME function instead of performing a -SHOWBACKLOG in a continuous loop. Please refer to the examples for more information on using ^%PEEKBYNAME for monitoring replication speed and backlog.

Example:

```
$ mupip replic -source -showbacklog -inst=INSTB
Tue Jan  5 10:36:46 2021 : Initiating SHOWBACKLOG operation on source server pid [0] for secondary instance [INSTB]
52 : 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
50 : sequence number of last transaction sent by source server
25 : sequence number acknowledged by the secondary instance [INSTB]
```

Example:

```
replspeed
  ; This routine returns the replication speed, that is the number of seqno updates per second acknowledged by the
▶ replicating instance during heartbeat intervals
  ; Usage: $gtm_exe/mumps -r ^replspeed INSTANCE3 20
  ; The second parameter is the sampling size
  set $etrap="write ""REPLSPEED-E-ACKSMPL : unable to fetch sampling data due to "",$zstatus  halt  "
  set $ztimeout="300:write ""Timeout occurred out after 5 minutes"",! zwrite  halt"
  new hrtbtperiod,instance,samplingsize,hrtbts,slot,i,hrtbtdiffs,diff,dump
  set instance=$piece($zcmdline," ",1),slot=0
  set samplingsize=$piece($zcmdline," ",2),hrtbtdiffs=0
  set:$length(samplingsize)=0 samplingsize=10
  if '($length(instance)) write "REPLSPEED-E-ARGS : ",$zcmdline," was specified. This routine requires specifying
 an
```

```
▶ instance name.",! halt
  for i=0:1:15 set
▶ instname=$$^%PEEKBYNAME("gtmsource_local_struct.secondary_instname",i),instname=$piece(instname,$char(0),1) set:
(instname=instance) slot=i
  ; capture heartbeat_jnl_seqno samplingsize times. Wait for hrtbtperiod after every capture of heartbeat_jnl_seqno
  set hrtbtperiod=$piece($$^%PEEKBYNAME("gtmsource_local_struct.connect_parms",slot),",",5)
  for j=1:1:samplingsize do
  . set hrtbts(j)=$$^%PEEKBYNAME("gtmsource_local_struct.heartbeat_jnl_seqno",slot,"I")
  . do:(j>1)
  . . set diff=hrtbts(j)-hrtbts(j-1)
  . . do:(diff<0)
  . . . write "REPLSPEED-E-ACKSEQNO : acknowledgement sequence number received is lower than previous
 acknowledgement
▶ seqno."
  . . . set dump="replspeed.dump" open dump use dump zwrite hrtbts  close dump
  . . . halt
  . . set hrtbtdiffs=hrtbtdiffs+diff
  . hang hrtbtperiod
  set hrtbtdiffs=hrtbtdiffs/samplingsize
  write $justify(hrtbtdiffs/hrtbtperiod,0,0)
  quit
```

▼

Example:

```
waitforhrtbt
  ; This routine returns 0 when the acknowledged seqno from the specified instance matches or exceeds the
 specified
▶ seqno
  ; If there is no confirmation (network issues etc) from the Receiver Server for 300 seconds, this routines
 returns 1.
  ; Usage: $gtm_exe/mumps -r ^waitforhrtbt <instname> <seqnocheckpoint>
  set $etrap="write ""WAITFORHRTBT-E-ERROR, Error occurred while waiting for ackseqno confirmation due to "",
$zstatus
▶ halt  "
  new heartbeatseqno,hangduration,instance,checkseqno,i,instname,slot
  ; set $ztimeout to align with the REPLALERT threshold for the test system
  set $ztimeout="900:write 1 halt"
  set hangduration=1,slot=""
  set instance=$piece($zcmdline," ",1)
  set checkseqno=$piece($zcmdline," ",2)
  if '($length(instance)&$length(checkseqno)) write "WAITFORHRTBT-E-ARGS : ",$zcmdline," was specified. This
▶ routine requires specifying instance name and seqno.",! halt
  for i=0:1:15 set
▶ instname=$$^%PEEKBYNAME("gtmsource_local_struct.secondary_instname",i),instname=$piece(instname,$char(0),1) set:
(instname=instance) slot=i
  if '($length(slot)) write "WAITFORHRTBT-E-INSTNOMATCH : No matching instance name for ",instance,! halt
  for  do
  . set heartbeatseqno=$$^%PEEKBYNAME("gtmsource_local_struct.heartbeat_jnl_seqno",slot,"I")-1
  . if (heartbeatseqno>=checkseqno) write 0 halt
  . hang hangduration
  quit
```

▼

311

Example:

```
waitforseqnosync
  ; This routine returns 0 when there the sequence numbers of the Source Server and the acknowledged sequence
 number
 from the Receiver Server is the same.
  ; It returns a non-zero value when there is no confirmation of the receipt of the latest seqno from the
 secondary
 even when there is no backlog.
  ; If there is no confirmation (network issues etc) from the Receiver Server for 150 seconds, this routines
 returns 1.
  ; Usage: $gtm_exe/mumps -r ^waitforseqnosync <instname>
  set $etrap="write ""WAITFORSEQNOSYNC-E-SRCBACKLOG : unable to get current Source Server backlog and seqno
 updates
 status due to "",$zstatus  halt  "
  new readseqno,heartbeatseqno,instance,i,instname,slot,hrtbtperiod
  set $ztimeout="150:write 1 halt"
  set slot=""
  ; hrtbtperiod: the heartbeat period (the fifth parameter of -CONNECTPARAMS)
  set instance=$zcmdline
  if '$length(instance) write "WAITFORSEQNOSYNC-E-ARGS : This routine requires specifying an instance name.",! halt
  for i=0:1:15 set
 instname=$$^%PEEKBYNAME("gtmsource_local_struct.secondary_instname",i),instname=$piece(instname,$char(0),1) set:
(instname=instance) slot=i
  if '($length(slot)) write "WAITFORSEQNOSYNC-E-INSTNOMATCH : No matching instance name for ",instance halt
  set hrtbtperiod=$piece($$^%PEEKBYNAME("gtmsource_local_struct.connect_parms",slot),",",5)
  for  do
  . set seqno=$$^%PEEKBYNAME("jnlpool_ctl_struct.jnl_seqno","","I")
  . set readseqno=$$^%PEEKBYNAME("gtmsource_local_struct.read_jnl_seqno",i,"I")
  . set heartbeatseqno=$$^%PEEKBYNAME("gtmsource_local_struct.heartbeat_jnl_seqno",i,"I")
  . if (seqno=readseqno=heartbeatseqno) write 0 halt
  . hang hrtbtperiod
  quit
```

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 -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 instnace 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.

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:

```
GDSJEX07 ROLLBACK SECONDARY Perth
```

Starting the Receiver Server

The MUPIP REPLICATE -RECEIVER -START command starts a Receiver Server and the Update Process. The general syntax for starting a Receiver Server and the Update Process is:

```
mupip replicate -receiver -start
[autorollback_qualifier]
connection_qualifiers
logfile_management_qualifiers
[operator_override_qualifiers]
[replication_filter_qualifers]
[si_replication_qualifiers]
[tls_replication_qualifier]
```

The square brackets [] denote an optional qualifier group. The optional and mandatory qualifiers in each qualifier group are as follows:

Category	Qualifiers
autorollback_qualifier	• **-autorollback[=verbose]**
connection_qualifiers	• **[-cmplvl=<compression_level>]**
	• **-listenport=<port number>**

Database Replication

Category	Qualifiers
logfile_management_qualifiers	• **-log=<log file name>** • **[-log_interval="[integer1],[integer2]"]**
operator_override_qualifiers	• **{-noresync | -updateresync=</path/to/bkup-orig-repl-inst-file>}**
replication_filter_qualifier	• **-filter=<filter command>**
recvpool_setup_qualifiers	• **-buffsize=<Receiver Pool size in bytes>**
si_replication_qualifiers	• **[-initialize]** • **[{-resume=<strm_num>|-reuse=<instname>}]**
tls_replication_qualifiers	• **-tlsid=<label>**

Autorollback Qualifier:

  -autorollback[=verbose]

-AUTOROLLBACK in a Receiver Server startup command of a BC/SI replication instance performs an automatic MUPIP JOURNAL -ROLLBACK -BACKWARD -FETCHRESYNC=<portno>. Choosing between -AUTOROLLBACK and -FETCHRESYNC (with -[NO]LOSTTRANS and -[NO]BROKENTRANS) depends on your replicating configuration and how your application processes lost transaction files. Use a MUPIP JOURNAL -ROLLBACK command with -FETCHRESYNC when you need:

- Control over the name and location of the lost/broken transaction files (by specifying -LOSTTRANS and -BROKENTRANS)

- To disable lost transaction file processing if there is no need to apply lost transaction files (by specifying -LOSTTRANS=/dev/null or -NOLOSTTRANS), or

- To disable broken transaction file processing if there is no need to research broken transaction (by specifying -BROKENTRANS=/dev/null or -NOBROKENTRANS), or

- To enable operational intervention when there is a lost transaction file in order to capture, review or process it on the new originating/primary instance before starting the Receiver Server.

For more information on MUPIP JOURNAL -ROLLBACK -BACKWARD -FETCHRESYNC, refer to "Rolling Back a Replicated Database " (page 321).

Use -AUTOROLLBACK when there are no application side restraints on the timing/need of processing of the lost transaction file. With -AUTOROLLBACK, the Receiver Server performs a connection handshake with the originating/upstream Source Server. If the upstream Source Server sends the REPL_ROLLBACK_FIRST message during the handshake, the Receiver Server with -AUTOROLLBACK performs the following operations:

- Close its connection with the Source Server.

- Uses the Source Server's connected port number stored in the memory to launch a separate MUPIP process for MUPIP JOURNAL -ROLLBACK -FETCHRESYNC which receives the rollback point (region sequence number) from the originating/upstream Source Server and rolls back the replicating instance to that rollback point and generates a lost transaction file in the default location.

- Once the rollback is complete, the Receiver Server re-establishes the replication connection and resumes receiving updates from the Source Server.

- A Receiver Server started without AUTOROLLBACK shuts down with the message **"Receiver was not started with -AUTOROLLBACK. Manual ROLLBACK required. Shutting down"**.

Connection Qualifiers:

  -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.

Logfile Management Qualifiers:

-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 the Receiver Server should process between updates to its log file. *integer2* specifies the number of transactions the Update Process should process between updates 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.

Operator Override Qualifiers:

  -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.

- 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.

When started with -updateresync, the receiver server continues trying connection attempts to the source server until it gets a successful update in response. Once an update goes through, subsequent connection restarts behave as if the receiver was started without -updateresync.

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 (<bckup_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 290), "Replacing the replication instance file of a replicating instance (A→B and P→Q)" (page 290), "Replacing the replication instance file of a replicating instance (A→P)" (page 291), and "Setting up a new replicating instance from a backup of the originating instance (A→P)" (page 291).

   -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 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 NORESYNCUPDATERONLY 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.

SI Replication Qualifiers:

   -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

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).

TLS Replication Qualifiers:

-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 gtmcrypt_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>=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* is not 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 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 the Receiver Server and/or the Update Process

Command syntax:

```
mupip replicate -receiver -shutdown [-helpers | -updateonly] [-timeout=<timeout in seconds>]
```

Qualifiers:

-shutdown

Initiates the shutdown procedures of the Receiver Server, Update Process, and/or helper processes. If the Receiver Server previously shut down abnormally, -shutdown can shut down helper processes left running.

-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 shutdown command should wait before signaling the Receiver Server, Update Process, and/or helper processes to shut down.If you do not specify -timeout, the default timeout period is 30 seconds. If you specify -timeout=0, shutdown occurs immediately.

-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.

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 [-he[lpers]]
```

With helpers specified, -checkhealth displays the status of Helper Processes in addition to the status of Receiver Server and Update Process.

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 a Replicated Database

Command syntax:

```
mupip journal -rollback
 {[-fetchresync=<port number>|-resync=<JNL_SEQNO>]
  -[no]losttrans=[/dev/null|<unreplicated_updates.lost>]
 -[no]brokentrans=[/dev/null|<brokentrans.broken>]
 -[no]online
  [-rsync_strm=<strm_num>]}
 -backward *
```

Qualifiers:

-rollback

Use -FETCHRESYNC or -RESYNC to rollback the database as a recovery step after a system failure or to make a change in the LMS replication configuration. If you do not use -FETCHRESYNC or -RESYNC, the database rolls back to the last consistent state.

-online

Specifies that ROLLBACK can run without requiring exclusive access to the database and the replication instance file. 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 -ROLLBACK -ONLINE, refer to "-ROLLBACK [{-ON[LINE]|-NOO[NLINE]}] " (page 210).

-fetchresync

When there are unreplicated updates on a former primary/secondary instance, it cannot become a new replicating instance as its journal sequence number is higher than the journal sequence number of its new replication source. Use -ROLLBACK -FETCHRESYNC to roll back a BC/SI replicating instance to a journal sequence number that matches the journal sequence number of its replication source and generate a lost transaction file containing the unreplicated updates for the organization to reconcile. After the reconcilation (or rejection as the case may be) of the lost transaction file, run MUPIP REPLICATE -SOURCE -LOSTTNCOMPLETE to provide confirmation to GT.M that you have applied the lost transaction file. For -FETCHRESYNC to work on the replicating instance, you need to ensure that an active Source Server is running on the replication source/ originating instance.

The format of the -FETCHRESYNC qualifier is:

-fetchresync=<port number> -[no]brokentrans=[/dev/null|<brokentrans.broken>] -[no]losttrans=[/dev/null| <unreplicated_updates.lost>]

- <portno> is the port number that the Receiver Server uses to listen for incoming connection from the Source Server of the originating instance.

- -LOSTTRANS and -BROKENTRANS allows you to specify the name and location of the lost transaction file and broken transaction file.

- -LOSTTRANS=/dev/null or -NOLOSTTRANS disables lost transaction file processing when there is no need to apply lost transaction files.

- Remember that unless you have a backup of the database and the journal files, you cannot undo a ROLLBACK/RECOVER operation.

- A lost transaction file may sometimes contain data that is critical for your application. Use -LOSTRANS=/dev/null or -NOLOSTTRANS with ROLLBACK/RECOVER only when you are sure that there is a lost transaction file whose transactions you purposely want to discard from your database.

- Unless business considerations dictate otherwise, your database administration scripts/procedure should allow the organization to intervene when there is a need to apply or discard a lost transaction file.

- If you have broken transactions, they may have appeared due to system failures that occurred during processing. Do not reconcile these transactions as they are not considered to be committed.

- In your Receiver Server startup scripts, FIS recommends placing MUPIP JOURNAL -ROLLBACK -BACKWARD -ONLINE - FETCHRESYNC after starting the passive Source Server but before starting the Receiver Server.

- Stopping a ROLLBACK operation before it completes or losing the TCP connection over which replication operates may corrupt the database file header. To recover, wait up to 60 seconds for the operation to timeout and then reissue the command.

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.

-rsync_strm=<strm_num>

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 V6.3-010	31 October 2019	• In "Clear" (page 326), remove non-existent -CRIT qualifier from LKE CLEAR
Revision V6.3-006	26 October 2018	• In "SHow " (page 330), add information about -NOCRIT and make information about the LOCKSPACEINFO up to date. • In "Clear" (page 326), made formatting adjustments and corrected the syntax of the CLEAR command. • In "CLNup" (page 329), add new section.
Revision V6.3-005	03 July 2018	• In "SHow " (page 330), specify that SHOW recognizes the full keyword for -CRITICAL. • In "Exercise 2: Rectifying a deadlock situation" (page 336), made formatting adjustments for the caution note.
Revision V6.0-000/1	21 November 2012	In "SHow " [330], added updated for V6.0-000.

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.

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:

```
CLE[AR] [-qualifier...]
```

The optional qualifiers are:

```
-A[LL]
-L[OCK]
-[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]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.

```
-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
```

*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
```

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.

CLNup

The **CLNUP** command initiates a cleanup operation of the lock space to remove any abandoned artifacts left by processes that exited without releasing their LOCKs.

The CLNUP processing also checks for the evidence of any entry that has been misplaced by an "overflow" condition; if it finds any, it attempts to automatically repair it, and, if it cannot, it produces a MLKHASHTABERR warning message. On seeing such a message:

1. Stop all access to (at least) the affected region(s) to ensure that the database is completely quiescent.

2. Use MUPIP SET -LOCK_SPACE to set, and, if appropriate raise, the number of pages allocated to the management of M locks associated with the affected region(s) before resuming normal operations.

Note that step 1 is necessary because using MUPIP SET -LOCK_SPACE is a standalone operation even with the current value.

The format of the CLNUP command is:

```
CLN[UP] [-qualifier...]
```

The optional qualifiers are:

```
-A[LL]
-I[NTEG]
-P[ERIODIC]=n
-R[EGION]=region-name
```

By default, CLNUP operates on all regions (-ALL).

Qualifiers of CLNUP

```
-A[LL]
```

Specifies all regions.

```
-I[NTEG]
```

Specifies that LKE should validate the integrity of the lock space and report any issues.

```
-P[ERIODIC]=n
```

Specifies that LKE perform a CLNUP every n seconds, which, if you desire active cleanup, is much lighter weight than repeated invocations of LKE from a shell script.

You can stop LKE CLNUP -PERIODIC with MUPIP STOP <pid>.

FIS recommends running LKE CLNUP -PERIODIC=n with a value of n that appears to prevent growth in the elements in the lock space as reported by LKE SHOW over substantial periods of time.

```
-R[EGION]
```

Specifies that LKE restricts CLNUP operations to a region.

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[RITICAL]
-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.

- All invocations of LKE SHOW include utilization information, in the form of available/total space, about shared subscript data space related to LOCK commands. This information appears as a message of the form: **%GTM-I-LOCKSPACEINFO, Region: <region_name>: processes on queue: 0/160; LOCK slots in use: lll/120; SUBSCRIPT slot bytes in use: ssss/5052**. Additionally, LKE SHOW also displays a LOCKSPACEUSE message. If the lock space is full, LKE SHOW 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.

- 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.

For example, **%GTM-I-LOCKSPACEUSE, Estimated free lock space: 50% of 40 pages** shows the amount of free space along with the number of pages configured for lock space

> **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[RITICAL]
```

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).

- **-NOCRIT** displays the PID of any process currently holding the LOCK critical section.

```
-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.

```
-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
```

```
^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[XIT]
```

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	-

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*.

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 V7.0-000	12 February 2021	• In "File Header Data Elements " (page 339), Update Maximum TN and Maximum TN Warn
Revision V6.3-013	30 June 2020	• In "File Header Data Elements " (page 339), update default and maximum journal buffer size
Revision V6.3-008	24 April 2019	• In "GDS Records " (page 347), Update breakdown of the record header fields. • In "GDS Blocks " (page 347), Update breakdown of field within the block header.
Revision V6.3-007	04 February 2019	• In "File Header Data Elements " (page 339), Add a missing word to the Maximum Record Size description
Revision V6.3-006	26 October 2018	• In "String Subscripts " (page 349), UTF-8 tweaks
Revision V6.3-005	03 July 2018	• In "Exercise 2: Rectifying a deadlock situation" (page 336), made formatting adjustments for the caution note.
Revision V6.3-001	20 March 2017	• In "File Header Data Elements " (page 339), removed references to OpenVMS and added the descriptions of WIP queue cache blocks, DB is auto-created, DB shares gvstats. LOCK crit sharing, AsyncIO
Revision V6.0-003/1	19 February 2014	In "File Header Data Elements " [339], updated the descriptions for V6.0-003.
Revision V6.0-003	27 January 2014	In "GDS Blocks " [347], corrected the description of block header fields.

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) 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.
Async IO	Whether the database file uses Asynchronous or Syncrhonous I/O. For additional information, see Chapter 4: "Global Directory Editor" (page 45).
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 45).
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.

Data Elements	Description
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.
DB is auto-created	Indicates whether the database file is automatically created.
DB shares gvstats	Indicates whether the database supports sharing of statistics.
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 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).
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	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). If the ALIGNSIZE is not a perfect power of 2, GT.M rounds it up to the nearest power of 2. The default and minimum value is 4096. The maximum value is 4194304 (=2 GigaBytes).

GT.M Database Structure(GDS)

Data Elements	Description
	A small alignsize can make for faster recover or rollback operations, but makes less efficient use of space in the journal file.
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	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. The default value for Journal AutoSwitchLimit is 8386560 & 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.
Journal Before imaging	Indicates whether or not before image journaling is allowed; DSE only reports this field if journaling is ENABLED or ON. Journal Before imaging can either be TRUE or FALSE.
Journal Buffer Size	The amount of memory allotted to buffer journal file updates. The default value is 2312. The minimum is 2307 and the maximum is 1Mi blocks which means that the maximum buffer you can set for your journal file output is 512MB. Larger journal buffers can improve 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	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. 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.
Journal Extension	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. 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. 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.
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	Indicates whether WRITE operation to a journal file commits directly to disk. The default value is FALSE. DSE only reports this field if journaling is ENABLED (or ON).

Data Elements	Description
Journal Yield Limit	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. The minimum Journal Yield Limit is 0, the maximum Journal Yield Limit is 2048. 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.
KILLs in progress	The sum of the number of processes currently cleaning up after multi-block KILLs and the number of Abandoned KILLs. **Note** Abandoned KILLs are associated with blocks incorrectly marked busy errors.
Last Bytestream Backup	The transaction number of the last transaction backed up with the MUPIP BACKUP -BYTESTREAM command.
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 shares DB critical section	Whether LOCK activity shares the resource manager for the database or has a separate and different critical section manager.
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 "Global Directory Editor" (page 45).
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 issues 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 V7 format, the default value of Maximum TN is 0xFFFFFFF803FFFFFF.

GT.M Database Structure(GDS)

Data Elements	Description
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 0xFFFFFFD813FFFFFF.
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 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. 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.
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.
WIP queue cache blocks	The number of blocks for which GT.M has issued writes that have not yet complete.
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 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

Data Elements	Description
	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 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%.

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.
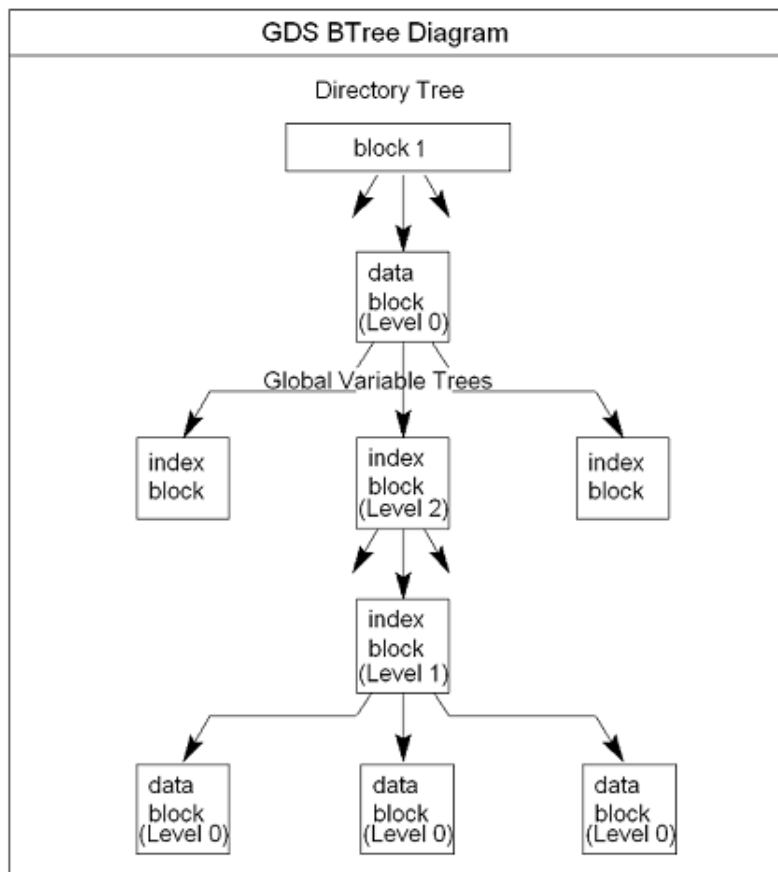
GT.M creates a new GVT when a SET results in the first use of an unsubscripted global name by referring to a subscripted or unsubscripted global variable with a name prefix that has not previously appeared in the database.

> **⚠ 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 one byte, specifies the block level. The third field, of four bytes, specifies the number of bytes currently in use in the block. The last field, of eight bytes, specifies 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
```

There is also be an empty field containing filler to produce proper alignment. The filler occurs between the first and second data field and causes the length of the header to increase from fifteen to sixteen 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**
>
> Depending on the platform an extra byte may be added to the compression count, allowing compression counts of up to 1020.

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 UTF-8 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|
          | .  .  .  . 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 V7.0-002	29 March 2022	• In "Examples for DUMP " (page 378), updated the -fileheader output
Revision V7.0-001	24 November 2021	• In "CHange " (page 361), correct Zqgblmod_T[rans]=sequence-number
		• In "DSE Command Summary" (page 395), correct Zqgblmod_T[rans]=sequence-number
		• In "Find " (page 381), Add the -ST[ATS] qualifier to the table
		• In "Qualifiers of FIND" (page 381), In this section, Add a -Stats qualifier information.
Revision V7.0-000	12 February 2021	• In "Examples for CHANGE" (page 370), fix dse command to update maximum record size, to pass just -record_max_size and not -reserved_bytes
		• In "DSE Command Summary" (page 395), correct the syntax of the -SIBLING qualifier
		• In "Qualifiers of FIND" (page 381), correct the syntax for the -SIBLING qualifier
Revision V6.3-014	06 October 2020	• In "Qualifiers of SHIFT" (page 393), remove spurious qualifier
Revision V6.3-013	30 June 2020	• In "DSE Command Summary" (page 395), fix typo: change HARD_SPIN_CPUNT to HARD_SPIN_COUNT and formatting cleanup.
Revision V6.3-007	04 February 2019	• In "Examples for DUMP " (page 378), update DSE dump -fileheader output for V6.3-007
Revision V6.3-003	12 December 2017	• In "CHange " (page 361), mark abbreviations
Revision V6.3-002	22 August 2017	• In "CHange " (page 361), fix typo in hard spin count
		• In "CHANGE -FIleheader Qualifiers" (page 364), add STDNULLCOLL, clarify the radixes of qualifiers with values, and remove duplicate entry for -TIMERS_PENDING
Revision V6.3-001	20 March 2017	• In "Qualifiers of CACHE" (page 373), added the missing command
		• In "ALL" (page 358), added information about the -DUMP and -ALL qualifiers.

		• In "CHange " (page 361), added SLEEP_SPIN_COUNT and SPIN_SLEEP_MASK. • In "CHANGE -FIleheader Qualifiers" (page 364), fix typo • In "DSE Command Summary" (page 395), added information about the -DUMP and -ALL qualifiers of ALL; add -ALL for DUMP. • In "Examples of ALL" (page 360), changed Examples of ADD to Examples of ALL and fix the WCINIT example to be ALL -WCINIT • In "Examples for DUMP " (page 378), update the DUMP -FILEHEADER output for V6.3-001
Revision V6.2-002	17 June 2015	• In "CHange " (page 361), added the [NO]EPOCHTAPER qualifier and removed the deprecated SPIN_SLEEP_TIME qualifier. • In "ALL" (page 358), added the DUMP qualifier. • In "CRitical " (page 374), added the ALL qualifer.
Revision V6.2-001	27 February 2015	• Updated "SAve " (page 392), "REStore " (page 391) , "REMove " (page 390) , "CAche " (page 373) for V6.2-000 enhancements. • In "Dump " (page 376), added the recommendation of using the ZWR format with UTF-8 mode.
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 [361], 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 [361], added the description of -QBDRUNDOWN and updated the description of -CORRUPT_FILE qualifier for V6.0-000.

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"
```

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 4 V6
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  0 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]
-D[UMP] -A[LL]
-[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

-ALL

Displays additional information on the database most of which is useful for FIS in diagnosing issues.

Meaningful only with: -D[UMP]

-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.

-[D]UMP

Displays fileheader information.

Compatible with: -A[LL]

-[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 115).

- 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.

- 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 curent 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 ALL

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> ALL -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_BLKS_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
-FU[LLY_UPGRADED]=boolean
-GV[STATSRESET]
-HA[RD_SPIN_COUNT]=Mutex-hard-spin-count
-HE[XLOCATION]=value
-INT[ERRUPTED_RECOV]=boolean
-JNL_YIELD_LIMIT=journal-yeild-limit
-KE[Y_MAX_SIZE]=key-max-size
-KI[LL_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
-SLEE[P_SPIN_COUNT]=mutex-sleep-spin-count
-SPIN[_SLEEP_MASK]=mutex-spin-sleep-mask
-STRM_NUM=stream-number STRM_REG_SEQNO=hexa
-TIM[ERS_PENDING]=integer
-TO[TAL_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_TN=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 decimal positive integer.

Use only with: -FILEHEADER

-BLK[_SIZE]=block_size

Changes the decimal block size field of the current file.

- 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 extrordinary 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 hexidecimal 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 priot versions, this can still be specified as -B_COMPREHENSIVE.

-D[ATABASE]=transaction_number

Changes the hexidecimal 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 hexidecimal 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 decimal 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 hexidecimal 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 (in decimal), GT.M puts it at that location.

Use only with: -FILEHEADER

-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 441).

> ⚠️ **Caution**
>
> An incorrect hash renders the database useless.

Use only with: -FILEHEADER

-[NO]EPOCHTAPER

Sets a flag that indicates whether or not epoch tapering should be done. The default value is -EPOCHTAPER.

For more information, refer to "Region Qualifiers" (page 71).
-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).

- 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 115) 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 a hexadecimal offset with the file header. If -VALUE is specified, GT.M puts it at that location.

Use only with: -FILEHEADER

-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 45).

Use only with: -FILEHEADER

-KI[LL_IN_PROG]=value

Changes the value of the KILLs in progress field. The value can be "NONE" or a positive decimal integer.

Use only with: -FILEHEADER

-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 71).

-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 45).

-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 230).

-RESYNC_S[EQNO]=sequence-number

In an LMS environment, this sets the hexidecimal value of the "Resync Seqno" field. For more information, refer to Chapter 7: "Database Replication" (page 230).

-RESYNC_T[N]=sequence-number

In an LMS environment, this sets the hexidecimal value ofthe "Resync transaction" field. For more information, refer to Chapter 7: "Database Replication" (page 230).

-SPIN_SLEEP_MASK]=hexadecimal-mask

Changes the hexadecimal Spin sleep time mask that controls the maximum time in nanoseconds the process sleeps on a sleep spin; zero (0), the default causes the process to just yield to the OS scheduler.

Use only with: -FILEHEADER

-SLEE[P_SPIN_COUNT]=integer

Changes the hexadecimal Mutex Sleep Spin Count that controls the number of times a process waiting on a shared resource (usually a database) suspends its activity after exhausting its Mutex Hard Spin Count and before enquing itself to be awakened by a process releasing the resource

Use only with: -FILEHEADER

-[NO]STD[NULLCOL]

Changes the collation of empty string ("NULL") subscripts for the database file. Although it is not the default, STDNULLCOLL is required with certain other characteristics, and highly recommended in any case. If you change this when there are existing "NULL" subscripts the results may be problematic. FIS recommends you establish this characteristic with GDE and load data with a consistent setting.

Use only with: -FILEHEADER

-STRM_NUM=stream-number -STRM_R[EG_SEQNO]=str_num's_region_sequence_number

Changes the hexadecimal values of 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

-TI[MERS_PENDING]=timers_pending

Sets a field that tracks the decimal 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

-TO[TAL_BLKS]=total_blocks

Changes the hexidecimal total blocks field of the current file. Use only with: -FILEHEADER

> ⚠️ **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 -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 -zqgblmod_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]
-SH[OW]
-VE[RIFY]
]
```

Qualifiers of CACHE

< / ▯- 9 @/ < ▯ ▯▯+ 6 6▯

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 CACHE -RECOVER only if a DSE CACHE -VERIFY commands reports the cache is "NOT clean".

-SH[OW]

Displays the cache data structure information. All values are in 8-byte hexadecimal form. If the database has encryption turned on, SHOW additionally displays an element that gives information about the encrypted global buffer section in shared memory.

-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.

Example:

```
DSE> CACHE -SHOW
File    /home/jdoe/node1/areg.dat
Region  AREG
Region AREG : Shared_memory       = 0x00002B6845040000
Region AREG :   node_local        = 0x0000000000000000
Region AREG :   critical          = 0x0000000000010000
Region AREG :   shmpool_buffer     = 0x0000000000023000
Region AREG :   lock_space        = 0x0000000000125000
Region AREG :   cache_queues_state = 0x000000000012A000
Region AREG :   cache_que_header   = 0x000000000012A030 : Numelems = 0x00000407 : Elemsize = 0x00000098
Region AREG :   cache_record       = 0x0000000000150458 : Numelems = 0x00000400 : Elemsize = 0x00000098
Region AREG :   global_buffer      = 0x0000000000177000 : Numelems = 0x00000400 : Elemsize = 0x00000400
Region AREG :   db_file_header     = 0x0000000000277000
Region AREG :   bt_que_header      = 0x00000000002B7000 : Numelems = 0x00000407 : Elemsize = 0x00000040
Region AREG :   th_base           = 0x00000000002C71D0
Region AREG :   bt_record         = 0x00000000002C7200 : Numelems = 0x00000400 : Elemsize = 0x00000040
Region AREG :   shared_memory_size = 0x00000000002D8000
DSE>
```

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]
[
-A[LL]
-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

-A[LL]

Display all ids of processes owning critical section from all regions. If there are no processes owning critical section in a region, ALL displays "the CRIT is currently unowned" message for each region.

-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 lightening 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.

Meaningful only with: -FILEHEADER

-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. [The GLO format is not supported for UTF-8 mode - use the ZWR format with UTF-8 mode.

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/V7.0-002_x86_64/g/gtm.dat
Region  DEFAULT
File            /home/jdoe/.fis-gtm/V7.0-002_x86_64/g/gtm.dat
Region          DEFAULT
Date/Time       23-MAR-2022 03:54:59 [$H = 66191,14099]
  Access method                     BG  Global Buffers              1024
  Reserved Bytes                     0  Block size (in bytes)       4096
  Maximum record size              256  Starting VBN                8193
  Maximum key size                  64  Total blocks       0x0000000000000065
  Null subscripts                NEVER  Free blocks        0x0000000000000062
```

```
 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
 Freeze online                     FALSE  Freeze online autorelease      FALSE
 Current transaction   0x0000000000000001  No. of writes/flush                7
 Maximum TN            0xFFFFFFFF803FFFFFF  Certified for Upgrade to          V7
 Maximum TN Warn       0xFFFFFFD813FFFFFF  Desired DB Format                 V7
 Master Bitmap Size                 8176  Blocks to Upgrade 0x0000000000000000
 Create in progress                FALSE  Modified cache blocks              0
 Reference count                       1  Wait Disk                          0
 Journal State           [inactive] ON  Journal Before imaging          TRUE
 Journal Allocation                 2048  Journal Extension               2048
 Journal Buffer Size                2312  Journal Alignsize               4096
 Journal AutoSwitchLimit         8386560  Journal Epoch Interval           300
 Journal Yield Limit                   8  Journal Sync IO                FALSE
 Journal File: /home/jdoe/.fis-gtm/V7.0-002_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
 Defer allocation                  TRUE  Spin sleep time mask      0x00000000
 Async IO                           OFF  WIP queue cache blocks             0
 DB is auto-created               FALSE  DB shares gvstats               TRUE
 LOCK shares DB critical section  FALSE  Read Only                        OFF
 Recover interrupted              FALSE  Full Block Write                   0
 StatsDB Allocation                2050
```

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 338).

EValuate

Translates a hexadecimal number to decimal, and vice versa.

The format of the EVALUATE command is:

```
EV[ALUATE]
```

```
[
-D[ECIMAL]
-H[EXADECIMAL]
-N[UMBER]=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[ECIMAL]

Specifies that the input number has a decimal base.

Incompatible with: -HEXADECIMAL .

-H[EXADECIMAL]

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]
-SI[BLINGS]
-ST[ATS]
]
```

- 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. FREEBLOCK accepts bit maps as starting or ending points.

- 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.

- DSE accepts ^#t as a valid global name when specifying a key.

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.

-SI[BLINGS]

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

-ST[ATS]

Switches to the named Global Directory shadow for the region's shared gvstats.

Compatible only with R[EGION].

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
```

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 commands 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[NTEGRIT] -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 121) 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. BUSY accepts bit map blocks.

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. MASTER accepts bit map blocks.

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: |  :XXXXXXX  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. DSE RANGE accept bit maps as starting or ending points.

By default, RANGE starts processing at the beginning of the file.

-T[O]=block-number

Specifies an ending block number for the range search. DSE RANGE accept bit maps as starting or ending points. 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).

BLOCK accepts blocks higher than the current database size because they deal with set of saved block copies rather than the database and there are situations where a saved block may be outside the current database size (for example, due to a concurrent MUPIP REORG -TRUNCATE).

-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 boundry).

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. If there are more than one version of a block, -VERSION is required; otherwise REMOVE works on that sole version. -VERSION is incompatible with all qualifiers except -BLOCK.

If there is only one version of the specified -BLOCK= block in the current region, DSE REMOVE defaults to that version.

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).

BLOCK accepts blocks higher than the current database size because it deal with set of saved block copies rather than the database and there are situations where a saved block may be outside the current database size (for example, due to a concurrent MUPIP REORG -TRUNCATE).

-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. If there is only one version of the specified -FROM= block, DSE RESTORE defaults to that version and it always restores the original block transaction number.

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. SAVE can preserve 128 versions. 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.

LIST may display blocks higher than the current database size because it deals with set of saved block copies rather than the database and there are situations where a saved block may be outside the current database size (for example, due to a concurrent MUPIP REORG -TRUNCATE);

Incompatible with: -COMMENT

-[NO]C[RIT]

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
-BL[OCK]=block_number
-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

-BL[OCK]=block_number

Specifies the block number to perform the DSE SHIFT.

-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

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]	-A[LL]	Meaningful only with -DUMP
-	-B[UFFER_FLUSH]	Incompatible with -RENEW
-	-C[RITINIT]	Incompatible with -RENEW, -RELEASE, -SEIZE
-	-D[UMP]	Use with: -ALL
-	-[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
CA[CHE]	-ALL	Used with -RECOVER, -SHOW, and -VERIFY
-	-RE[COVER]	Use only with -ALL.
-	-SH[OW]	Use only with -ALL.
-	-VE[RIFY]	Use only with -ALL.
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

COMMAND	QUALIFIERS	COMMENTS
-	-OF[FSET]=offset	Use only with -BLOCK, -CMPC, -RSIZ
-	-RE[CORD]=record number	Use only with -BLOCK, -CMPC, -RSIZ
-	-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_BLKS_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_COUNT=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	

COMMAND	QUALIFIERS	COMMENTS
-	-N[ULL_SUBSCRIPTS]=value	Use only with -FILEHEADER
-	-NO[CRIT]	
-	-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=mutex sleep spin count	Use only with -FILEHEADER;
-	-SPIN_SLEEP_TIME=mutex 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_TN=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

COMMAND	QUALIFIERS	COMMENTS
D[UMP]	-B[LOCK]=block_number	Incompatible with -FILEHEADER
-	-C[OUNT]=count	Incompatible with -FILEHEADER
-	-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
-	-SI[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	

COMMAND	QUALIFIERS	COMMENTS
-	-O[FFSET]=offset	-
P[AGE]	-	-
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 V7.0-001	24 November 2021	• In "DSE Command Summary" (page 395), correct Zqgblmod_T[rans]=sequence-number
Revision V7.0-000	12 February 2021	• In "DSE Command Summary" (page 395), correct the syntax of the -SIBLING qualifier
Revision V6.3-013	30 June 2020	• In "DSE Command Summary" (page 395), fix typo: change HARD_SPIN_CPUNT to HARD_SPIN_COUNT and formatting cleanup.
Revision V6.3-010	31 October 2019	• In "MUPIP INTEG Error Classification Table" (page 421), add more explanation • In "Run-Time Database Restart Codes" (page 437), fix section title • In "C1–Possible Cache Control Problems" (page 412), add a link to section R3 • In "Can Be Deferred" (page 420), update MUPIP INTEG error messages table and give it its own section • In "Finding and Fixing Database Errors" (page 412), separate mupip integ error table and restart code table into a separate sections • In "H1–Process Hangs" (page 413), improve the instructions • In "H3–Database Access Problems" (page 414), modify some instructions • In "H7–Disk Hardware Problems" (page 417), remove inappropriate text • In "I1–MUPIP INTEG Errors" (page 420), put table in a separate section • In "I3–File Header Errors" (page 425), add mupip dumpfhead as a reporter of freecnt • In "I5–More Database Access Problems" (page 425), remove inappropriate text • In "K5–Star Key Problems" (page 428), more accurate wording • In "I6–Transient Errors" (page 426), add discussion of the role of startup procedures. • In "I7–Database Rundown Problem" (page 426), modify the description to reflect the use of appropriate startup.

		• In "K3–Block Doubly Allocated" (page 428), improve wording
		• In "M1–Bitmap Errors" (page 429), enhance wording of MAP -MASTER
		• In "O3–Data Block Errors" (page 430), additional recommendations
		• In "O4–Salvage of Data Blocks with Lost Indices" (page 431), adjust wording
		• In "P1–Process Damage" (page 435), enhance wording
		• In "R2–Structural Database Integrity Errors" (page 437), additions and adjustments
		• In "Repairing the Database with DSE" (page 405), additional instructions
Revision V6.3-006	26 October 2018	• In "P1–Process Damage" (page 435), add clarification.
Revision V6.3-005	03 July 2018	• In "R2–Structural Database Integrity Errors" (page 437), revise wording for H
Revision V6.3-004	23 March 2018	• In "Download salvage.m" (page 431), add section on downloading salvage.m
Revision V6.3-001	20 March 2017	• In "DSE Command Summary" (page 395), added information about the -DUMP and -ALL qualifiers of ALL; add -ALL for DUMP.
		• In "H7–Disk Hardware Problems" (page 417), replaced OpenVMS with OS.

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

- 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

Doing repairs with DSE should only be necessary if things have gone very wrong. MUPIP ROLLBACK and RECOVER are much better options in the vast majority of situations.

When using DSE:

- Always work in pairs to ensure appropriate planning and risk minimization.

- Restrict concurrent activity at least to the portion of the database on which you are working - for small changes, use CRIT -SEIZE to temporarily suspend database updates and CRIT -RELEASE to allow database updates.

- If at all possible, take a backup of the database before starting.

- Before changing a block, do a save and/or dump to a file - remember saves are in memory and lost on exit from DSE

- Address bit maps "incorrectly free busy" first - they are worse trouble about to happen

- Address bit maps "incorrectly marked busy" last - they are locators for potentially important data in jeopordy

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 MUPIP 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.

  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.

- 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.

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 : tttttt 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.

1. Assume that during the load of a binary extract, you get the following error:

```
%GTM-E-LDSPANGLOINCMP, 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-LDSPANGLOINCMP, 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–Process Hangs" [413].

IF THE SYMPTOM IS A MUPIP INTEG ERROR REPORT, refer to section "I1–MUPIP INTEG Errors" [420]. If you are investigating a particular error message, refer to the "MUPIP INTEG errors" table in that section.

IF THE SYMPTOM IS A GT.M RUN-TIME ERROR REPORT, refer to section R1. Remember the GT.M Message and Recovery Procedures Manual may provide insights.

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.

Also see section "R3–Run-time Database Cache Problems" [439].

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 H3 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 H?.

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?

  - 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?

- Do MUTEXLCKALERT messages indicate a process is holding the critical section for one or more regions?

  - If so, and there are more than one such process, examine the MUTEXLCKALERT messages to see if there is a deadlock cycle. GT.M is designed to prevent such a situation, but you should check. Cycles involve processes owning resources needed by another; typical cycles involve two processes, but can involve more.

  - Do any persistant critical section owning processes respond to a MUPIP INTRPT, which may provide important diagnostic information?

- Do any persistant critical section owning processes respond to a MUPIP STOP?

- If critical section owning processes do not resond to a MUPIP STOP but all database activity on the held region(s) is at standstill, does one or more respond to a kill -9? FIS does not recommend using a kill -9 on a process that is currently performing, or may perform database updates; however in this case you have determined all activity has stopped.

- If the MUPIP STOP or unmentionalble alternative just makes another process assume control of the critical sections without eliminating the hang, you can try a CRIT -INIT -RESET

- Does a CRIT -INIT -RESET free the critical section or just change who owns it?

- If CRIT -INIT -RESET doesn't free the problem, the global buffer pool management is damaged.

The following is another way of testing the cache: If CRIT is cleared and DSE BUFFER hangs, management of the global buffer pool is not working properly. 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 a safe dummy node in each database and then KILL it if appropriate. Alternatively make a harmless modification to an existing node, or modify a node and then change it back to its original value. GT.M detects when a SET makes no actual change (the SET value is the same as the original value) and does not perform an actual update in that case.

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:

```
 Set pipe="pipe"
 Open pipe:(command="/bin/csh")::pipe
 Use pipe
 Set reg="",cmd=$ztrnlnm("gtm_dist")_"/mupip dumpfhead "
 For  Set reg=$View("GVNEXT",reg) Quit:""=reg  Do
 . Set reg(reg)="",file=$view("GVFILE",reg)
 . Write cmd,file,!
 . For i=1:1 read x(i):1 Quit:(x(i)["sgmnt_data.freeze")!$ZEOF!'$Test
 . Set pid=+$Piece(x(i),"=",2)
 . Set:pid frozen(reg)=pid
 Close pipe
 Set g="^%",$etrap="Write $ZStatus Set $ecode="""" Quit"
 Write !,"Attempting read access"
 If $Data(^%) Set reg=$View("REGION",g) Do read1
 For  Set g=$Order(@g) Quit:""=g  Set reg=$View("REGION",g)  Do:""=(reg(reg)) read1
 Set reg=""
 Write !!,"Attempting write access"
 For  Set reg=$Order(reg(reg)) Quit:""=reg  Do write1
 Write !
 Quit
read1
 Write !,"Read in region: ",reg," of ",g," successful"
 If ($Data(@g)#2) Set reg(reg)=g
 Else  Set reg(reg)=$Query(@g)
 Quit
write1
```

```
 If $Data(frozen(reg)) Write !,"Region ",reg," Frozen by PID ",frozen(reg) Quit
 If ""=reg(reg) Write !,"Region ",reg," has no data" Quit
 Write !,"Write to region: ",reg
 Set x=$Get(@reg(reg),"Yndef")
 Set @reg(reg)=1,@reg(reg)=x
 If "Yndef"=x ZKill @ref(ref); assumption that a value of Yndef is very unlikely
 Write " of ",reg(reg)," successful"
 Quit
```

This routine provides a generalized approach to automating some of the tasks described in this section. The routine issues a report if a region is frozen or completely empty of data. 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 routine can be extended by having it cycle through a set of Global Directories should that be appropriate.

> ### 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 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 (that is, 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.

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.

- 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.

MUPIP INTEG Error Classification Table

The following list of INTEG messages classifies error severity using the "nature" codes, and refers you to a section identifying an appropriate follow-up action.

• *A-Access*: prevents database access - these indicate an operational error or a fundamental file damage that prevents INTEG from providing more information.

• *B-Benign*: presents no risk of additional damage and has little or no effect on database performance; many bit map errors come under this nature code, but "incorrectly marked free errors" are dangerous.

• *D-Dangerous*: presents a high risk in the sense that continuing updates may cause significant additional damage

• *I-Index*: if the block is an index block, continuing updates will be very dangerous: INTEG reports such errors with a nature of DANGER; if the block is a data block, reported by INTEG with a nature of DATA, continuing updates can only cause limited additional damage, but some data may be lost, become inaccessible, or cause processes to inappropriately loop

• *S-Spanning*: prevents access to a block spanning node value; INTEG reports these with a nature of DATA

• *T-Transient*: usually cleared by an update to the database or possibly a straightforward operator action

INTEG reports these codes for many of the errors and in so doing transforms Index errors on index block to DANGER and otherwise to Data, meaning the issue is confined to a level 0 data block, and so very localized. However, when an index block has damage, GT.M cannot correctly navigate the tree and if operations continue, subsequent updates can go where they do not belong, causing increasing damage. When a data block has damage, the worst thing that can happen from a GT.M standpoint is that you get an indefinite loop. More commonly, some confined set of nodes becomes inaccessible, which may or may not be important from an application perspective, most commonly the application gets an error when it tries to use the data in question. It is possible for a single issue to cause multiple reports, and, in such a case, focus first on the most serious report.

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			
NATURE	MNEMONIC	ERROR MESSAGE	SECTION
T	BUFFLUFAILED	Error flushing buffers from rrrr for database file ffff.	I7
B	DBBADFREEBLKCTR	Free blocks counter in file header: nnnn appears incorrect, should be mmmm.	I3
I	DBBADKYNM	Bad key name.	K1
I	DBBADNSUB	Bad numeric subscript.	K1
D	DBBADPNTR	Bad pointer value in directory.	K4
I	DBBDBALLOC	Block doubly allocated.	K3
D	DBBFSTAT	Block busy/free status unknown (local bitmap corrupted).	M1
D	DBBNPNTR	Bit map block number as pointer.	K4
D	DBBPLMGT2K	Blocks per local map is greater than 2K.	I3

MUPIP INTEG Error Messages			
NATURE	MNEMONIC	ERROR MESSAGE	SECTION
D	DBBPLMLT512	Blocks per local map is less than 512.	I3
D	DBBPLNOT512	Blocks per local map is not a multiple of 512.	I3
I	DBBSIZMN	Block too small.	O1
I	DBBSIZMX	Block larger than file block size.	O1
A	DBBSIZZRO	Block size equals zero.	I3
T	DBBTUWRNG	The blocks-to-upgrade file-header field is incorrect. Expected nnnn, found mmmm.	H2
I	DBCMPBAD	Compression count not maximal.	K6
I	DBCMPNZRO	First record of block has nonzero compression count.	O1
I	DBCOMPTOOLRG	Record has too large compression count.	O2
A	DBCREINCOMP	Header indicates database file creation was interrupted before completion.	I3
I	DBDATAMX	Record too large.	O2
T	DBFGTBC	File size larger than block count would indicate.	I4
A	DBFLCORRP	Header indicates database file is corrupt.	I8
D	DBFSTBC	File size smaller than block count would indicate.	I4
A	DBFSTHEAD	File smaller than database header.	I3
I	DBGTDBMAX	Key larger than database maximum.	K7
A	DBHEADINV	Header size not valid for database.	I3
D	DBINCLVL	Block at incorrect level.	O1
A	DBINCRVER	Incorrect version of GT.M database.	I2
I	DBINVGBL	Invalid mixing of global names.	K3
I	DBKEYGTIND	Key greater than index key.	K2
I	DBKEYMN	Key too short.	K1
I	DBKEYMX	Key too long.	K1
I	DBKEYORD	Keys out of order.	K2
I	DBKGTALLW	Key larger than maximum allowed length.	K1
B	DBLOCMBINC	Local bit map incorrect.	M2
D	DBLRCINVSZ	Last record of block has invalid size.	K5
I	DBLTSIBL	Keys less than sibling's index key.	K2

MUPIP INTEG Error Messages			
NATURE	MNEMONIC	ERROR MESSAGE	SECTION
B	DBLVLINC	Local bitmap block level incorrect.	M1
I	DBMAXNRSUBS	Maximum number of subscripts exceeded.	K1
B	DBMBMINCFRE	Master bit map incorrectly asserts this local map has free space.	M1
B	DBMBPFLDIS	Master bit map shows this map full, in disagreement with both disk and INTEG result.	M1
B	DBMBPFLDLBM	Master bit map shows this map full, agreeing with disk local map.	M1
B	DBMBPFLINT	Master bit map shows this map full, agreeing with MUPIP INTEG.	M1
B	DBMBPFRDLBM	Master bit map shows this map has space, agreeing with disk local map.	M1
B	DBMBPFRINT	Master bit map shows this map has space, agreeing with MUPIP INTEG.	M1
B	DBMBPINCFL	Master bit map incorrectly marks this local map full.	M1
B	DBMBSIZMN	Map block too small.	M2
B	DBMBSIZMX	Map block too large.	M2
T	DBMBTNSIZMX	Map block transaction number too large.	I6
B	DBMRKBUSY	Block incorrectly marked busy.	M1
D	DBMRKFREE	Block incorrectly marked free.	M1
B	DBNONUMSUBS	Key contains a numeric form of subscript in a global defined to collate all subscripts as strings.	K1
A	DBNOREGION	None of the database regions accessible.	I6
A	DBNOTGDS	Unrecognized database file format.	I3
A	DBNOTMLTP	Block size not a multiple of 512 bytes.	K1
I	DBNULCOL	NULL collation representation differs from the database file header setting.	K1
D	DBPTRMX	Block pointer larger than file maximum.	K4
D	DBPTRNOTPOS	Block pointer negative.	K4
D	DBRBNLBMN	Root block number is a local bit tmap number.	K4
D	DBRBNNEG	Root block number negative.	K4
D	DBRBNTOOLRG	Root block number greater than last block number in file.	K4
T	DBRDONLY	Database file ffff read only.	I6
D	DBREADBM	Read error on bit map.	H7

MUPIP INTEG Error Messages			
NATURE	MNEMONIC	ERROR MESSAGE	SECTION
D	DBRLEVLTONE	Root level less than one.	O1
D	DBRLEVTOOHI	Root level higher than maximum.	O1
I	DBRSIZMN	Physical record too small.	O2
I	DBRSIZMX	Physical record too large.	O2
S	DBSPANCHUNKORD	Chunk of nnnn blocks is out of order.	O5
S	DBSPANGLOINCMP	Spanning node is missing. Block no nnnn of spanning node is missing.	O5
D	DBSTARCMP	Last record of block has nonzero compression count.	K5
A	DBSVBNMIN	Start VBN smaller than possible.	I3
A	DBSZGT64K	Block size is greater than 64K.	I4
T	DBTNLTCTN	Current tn and early tn are not equal.	I6
T	DBTNNEQ	Cannot reset transaction number for this region.	I4
T	DBTNRESET	Transaction numbers greater than the current transaction were found.	I6
A	DBTNTOOLG	Total blocks equal zero.	I6
T	DBTTLBLK0	Block transaction number too large.	I6
A	DBUNDACCMT	Cannot determine access method; trying with BG.	I6
A	FREEZE	Database for region rrrr is already frozen, not INTEGing	I6
A	MUSTANDALONE	Could not get exclusive access to rrrr.	I6
B	NOGTCMDB	INTEG does not support operation on GT.CM database region.	I5
I	NULSUBSC	Null subscripts are not allowed for database file: rrrr.	K1
A	REGFILENOTFOUND	Database file ffff corresponding to region rrrr cannot be found.	I6

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), MUPIP DUMPFHEAD and DSE 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:

```
((file size - starting VBN + 1) / (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.

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, replace it, 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 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, 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 HHHHHHH"

- Run the following command:

  dse change -fileheader -current_tn=<HHHHHHH+1>

  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 HHHHHHH 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 startup and shutdown procedures to ensure a controlled shutdown. Startup procedures should use journaling and/or replication to recover from an unscheduled 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=MM to correct the database.

I7–Database Rundown Problem

A MUPIP INTEG may be performed without write access to the database file. However, in the case where the database file was improperly closed, it must be put in an appropriate state with a MUPIP JOURNAL -ROLLBACK/RECOVER or MUPIP RUNDOWN prior to being INTEGed.

I8–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 might 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 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=8. 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 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. Before beginning any MAP - MASTER operation, first ensure that the database has no active updaters and that there are no non-bitmap errors to resolve.

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. Use application and business process knowledge to research an if appropriate reconstruct lost data; remember, if recently processed, it may exist in journal files.

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 YOU HAVE A DUPLICATE NODE PROBLEM, 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 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 ZWR. 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.

Download salvage.m

salvage.m is a utility that removes all incorrectly marked busy blocks from the specified region. During execution it displays the DSE commands that it will execute and aborts execution when it encounters an error. It dumps the zwrite formatted content of blocks incorrectly marked busy to a file called <region>_db.zwr. Upon completion, it sets the abandoned_kills and kill_in_prog flags in the database fileheader to false. Click on to download salvage.m program. You can also download *salvage.m* from http://tinco.pair.com/bhaskar/gtm/doc/books/ao/UNIX_manual/downloadables/salvage.m.

Steps to run the salvage utility are as follows:

1. Perform an argumentless MUPIP RUNDOWN before running this utility.

2. Ensure that there are no INTEG errors other than the incorrectly marked busy block errors.

3. Run **$gtm_dist/mumps -r ^salvage**.

4. Specify the region name. If no region is specified, the utility assumes DEFAULT.

5. If the utility reports a DSE error, fix that error and run the salvage utility again.

After completing repairs with the salvage utility, open the <REGION>_db.zwr file and examine its contents. If there is a need to recover the data from the incorrectly marked busy blocks, perform a MUPIP LOAD <REGION>_db.zwr to load that data back to the database.

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=^mypoem(#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=^mypoem(#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=^mypoem(#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=^mypoem(#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=mypoem.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 multiple processes exhibit the same symptoms, the problem is likely embedded in the application code and/or the database.

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
.

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.

Run-Time Error Messages Identifying Potential System Problems		
ERROR MNEMONIC	**ERROR MESSAGE TEXT**	**SECTION**
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

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.

Run-Time Database Restart Codes

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 Restart 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
F	-	Not currently used.	-
* In the last retry may indicate a process problem			

Run-Time Database Restart Codes			
FAIL CODE	**RUN INTEG**	**DESCRIPTION**	**SECTION**
G	x	Cache record modified while in use by the transaction.	R3
H	x	Development of a new version of a block encountered a likely concurrency conflict.	R3
I	x	Level on a child does not show it to be a direct descendent of its parent.	O2
J	x	Block requested outside of file.	O2
K	-	Cache control problem encountered or suspected.	R3
L	-	Conflicting update of a block took priority.	R3
M	x	Error during commit that the database logic does not handle.	P1
N	-	Not currently used.	-
O	-	Before image was lost prior to its transfer to the journal buffer.	R3
P	-	Not currently used.	-
Q	x	Shared memory interlock failed.	R7
R	x	Critical section reset (probably by DSE).	R5
S	-	Attempt to increase the level beyond current maximum.	R8
T	-	Commit blocked by flush.	R3
U	x	Cache record unstable while in use by the transaction.	R3
V	-	Read-only process could not find room to work.	R9
W	-	Not currently used.	-
X	x	Bitmap block header invalid.	M2
Y	x	Record offset outside of block bounds.	O2
Z	x	Block did not contain record predicted by the index.	O2
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	-	Not currently used.	-
e	x	Attempt to read a block outside the bounds of the database.	R3
f	-	Conflicting update took priority on a non-isolated global and a block split requires a TP_RESTART.	R3
g	-		
***** In the last retry may indicate a process problem			

Run-Time Database Restart Codes			
FAIL CODE	**RUN INTEG**	**DESCRIPTION**	**SECTION**
h	-	The number of conflicting updates on non-isolated global nodes exceed an acceptable level and requires a TP_RESTART.	R3
i	-	Journal state or before image changed.	R3
		Not currently used.	-
j	-	Backup or integ lost before image block.	R3
k	-	Cache control state mismatch.	R3
l	-	Phase two commit held off block acquisition.	P1
m	-	Inhibit KILLs held off commit.	P1
n	-	Trigger definition changed during trigger processing.	P1
o	-	Online rollback requires a restart.	R3
p	-	Online rollback requires a restart.	R3
q	-	Tentative block removed by MUPIP REORG -TRUNCATE.	R3
r	-	Root block moved by MUPIP REORG.	R3
s	-	Instance freeze blocked commit.	H3
y	x	Root block unreliable.	K4
z	-	Currently not used.	-
* In the last retry may indicate 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 V7.0-003	24 June 2022	• In "Tested Reference Implementations " (page 465), Drop RHEL 6 and update other distros
Revision V7.0-002	29 March 2022	• In "Special note - GNU Privacy Guard and Agents" (page 467), Advise kill and restart of any existing GPG agent processes so that updates are recognized. • In "Using the reference implementation's custom pinentry program" (page 467), Correct typo. Should be GPG 2.2.4 not GPG 2.2.24
Revision V7.0-001	24 November 2021	• In "Device IO " (page 446), Add reference to the Programmer's Guide for data in motion
Revision V6.3-007	04 February 2019	• In "Encrypted Database Creation " (page 473), remove redundant information. Add a summary for encrypting the database. • In "Key Ring on Disk " (page 450), rewrite the section with corrections and improvements. • In "Master Key Configuration File and Encryption Keys " (page 452), remove redundant information • In "Changing the Encryption Keys " (page 473), remove redundant information and add a link to REORG -ENCRYPT. • In "Packaging " (page 474), remove the reference to BLOWFISH algorithm. • In "Installation " (page 469), remove the reference to libgtmcrypt_openssl_BLOWFISHCFB.so • In "Keys in the Process Address Space / Environment" (page 444), add a point about the importance of password management procedure and a note on using the GT.M restriction facility. • In "Long Lived Keys " (page 445), wording revision on key changes to better reflect the reality • In "Using the reference implementation's custom pinentry program" (page 467), add an introduction paragraph, corrections, and a reference to the manpages of gpg and gpg-agent.

Revision V6.3-005	03 July 2018	• In "Plugin Architecture & Interface " (page 473), fix a typo (change "provided compiled" to "provided")
Revision V6.3-004	23 March 2018	• In "Special note - GNU Privacy Guard and Agents" (page 467), GPG agent bug affecting 2.1.15 through 2.2.23 • In "Using the reference implementation's custom pinentry program" (page 467), GPG agent bug affecting 2.1.15 through 2.2.23
Revision V6.3-001	20 March 2017	• In "Key Ring on Disk " (page 450), gtm_dbkeys to gtmcrypt_config • In "Master Key Configuration File and Encryption Keys " (page 452), gtm_dbkeys to gtmcrypt_config • In "Plugin Architecture & Interface " (page 473), specified that reference implementation is not shipped in binary form • In "GT.CM " (page 446), specified that replication can be encrypted • In "Tested Reference Implementations " (page 465), specified that the reference impl is source only; remove Solaris and HP-UX. Changed the supported versions table to drop 12.04 and SLES. Also added newer Linux versions that we test with and cleanup up the blowfish discussion • In "Using the Reference Implementation with Older Releases " (page 477), corrected typos and tweaked wording • In "Packaging " (page 474), specified that reference implementation is not shipped in binary form • In "Warning " (page 448), specified that the reference implementation is source only. • In "Overview " (page 448), gtm_dbkeys to gtmcrypt_config • In "Overview " (page 443), specified that the reference impl is source only. • In "FIPS Mode" (page 447), fixed a typo and added a link to http://www.openssl.org/docs/fips; remove HP-UX and Solaris • In "Installation " (page 469), specified that the reference implementation is source only. • In "Special note - GNU Privacy Guard and Agents" (page 467), Updated for changes in the FIS recommendations for how GT.M works with a GPG agent.

		• In "Using the reference implementation's custom pinentry program" (page 467), use-standard-socket doc add-on
Revision V6.0-001	27 February 2013	Updated "Tested Reference Implementations " (page 465), "Installation " (page 469), and "Plugin Architecture & Interface " (page 473) for V6.0-001.

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 only the source 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 compile and use it as distributed, but please read and understand the section "Disclaimer " (page 443). 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 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[1].

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 XECUTE'd, those users can find ways to view and capture keys and passphrases. Note 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. Consider using the GT.M restriction facility to restrict the access to GT.M facilities which can be used to obtain a shell prompt or the GT.M direct mode prompt. For more information on configuring the GT.M restriction facility, refer to "Configuring the Restriction facility" (page 38).

3. If you forget the passphrase, there is no way to decrypt the data from the encrypted regions of a database. Therefore, ensure that you have secure password management procedures to handle password storage and retrieval of the keyring passphrase.

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, it may not be feasible to change them 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.[2]

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 " [446], 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 for sequential disk files or SOCKET devices, please consult the GT.M Programmer's Guide. Alternatively, you can call encryption routines from GT.M using the external call interface.

GT.CM

GT.M encrypts does not encrypt 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.

When a GT.CM 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 $gtmcrypt_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

Before using FIPS mode on these platforms, ensure that your OpenSSL or Libgcrypt installation provides a validated FIPS 140-2 implementation (see http://www.openssl.org/docs/fips/).

> **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.
Key management	The generation, distribution, and access of keys. The reference implementation of database encryption uses: 1. symmetric keys to encrypt data and index records.

Terms	Description
	2. public keys to encrypt symmetric keys (so they can be placed on disk).
	3. private keys to decrypt symmetric keys.
	4. passwords to encrypt private keys (so they can be placed on disk).
Master key file	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.
Obfuscation	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.
Password (or Passphrase)	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).
Public key / Private key (or Asymmetric keys)	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). 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 libgpgme and libgpg-error for asymmetric key encryption.
Symmetric key	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.

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 source 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.
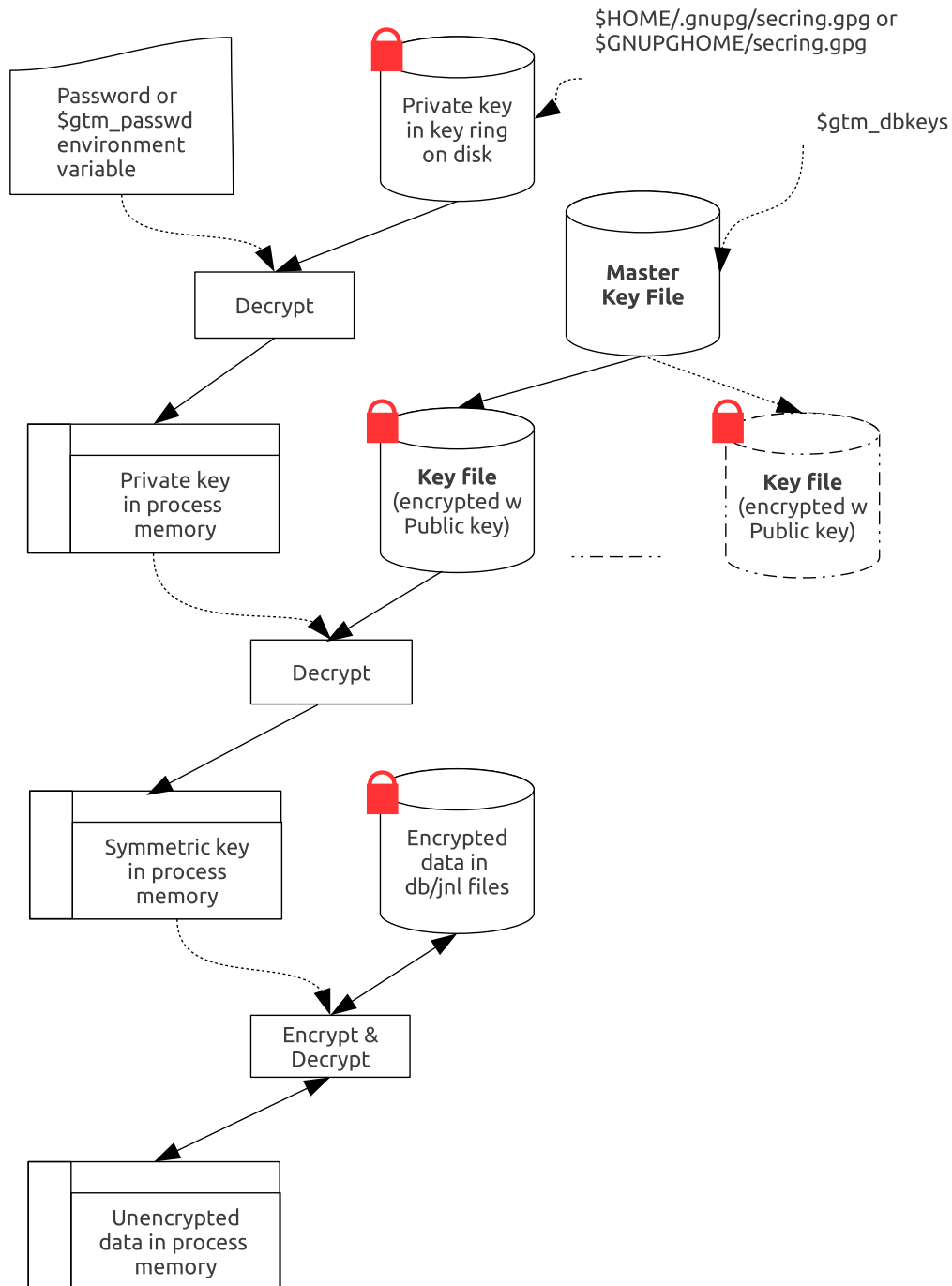
Key Ring on Disk

A passphrase protects the key ring on disk that contains the private key uses to encrypt the asymmetric database keys. GT.M requires this passphrase either in an obfuscated form as the value of the gtm_passwd environment variable, or,if gtm_passwd is set to "" and then typed at the GTMCRYPT passphrase prompt. GT.M obfuscates the passphrase to prevent inadvertent

disclosure, for example, in a dump of the environment that you may submit to FIS for product support purposes, the passphrase 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.

You can provide the passphrase of the key ring to GT.M in one of the following four ways:

- Set the gtm_passwd environment variable to an obfusacted form of the passphrase of the keyring using the maskpass utility but do not define $gtm_obfusaction_key. For example:

```
echo -n "Enter keyring passphrase: " ; export gtm_passwd=`$gtm_dist/plugin/gtmcrypt/maskpass|cut -f 3 -d "
▶ "`
```

▼

You should use this method when you need to restrict the access of encrypted regions to the same $USER using the same GT.M distribution. $gtm_passwd can be passed between from the parent process to a child process with the Job command. Note that $gtm_passwd is the only way for a child process to receive a password from a parent process.

- Set the environment variable gtm_passwd to "". In this case, GT.M uses the default GTMCRYPT passphrase prompt to obtain a password at process startup and uses that value as $gtm_passwd for the duration of the process. Note that you cannot change the GTMCRYPT passphrase prompt without customizing the reference implementation plugin.

- When the environment variable gtm_passwd is not set, create a one line GT.M program as in the following example:

```
echo 'zcmd ZSYstem $ZCMdline  Quit' > zcmd.m
```

and use it invoke the MUPIP or DSE command. For example:

```
$ gtm_passwd="" mumps -run zcmd mupip backup \"*\"
```

The empty string value of $gtm_passwd causes the GT.M 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.

> **Note**
>
> 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".

- Set the gtm_obfusaction_key environment variable to the absolution location of a file having any contents and then set the environment variable gtm_passwd to an obfusacted form of the passphrase of the keyring using the maskpass utility. **maskpass** is a stand-alone program that takes the passphrase from STDIN and writes its obfuscated value in the form of **Enter Passphrase: <obfuscated_value>** as its output. The <obfuscated_value> can then be set for the environment variable gtm_passwd. For example:

```
export gtm_obfuscation_key="/path/to/secret_content"
echo -n "Enter keyring passphrase: " ; export gtm_passwd=`$gtm_dist/plugin/gtmcrypt/maskpass|cut -f 3 -d "
▶ "`
```

▼

The maskpass utility uses the hash of the contents of the $gtm_obfuscation_key file to obfuscate the passphrase. You should use this method when you need to allow multiple users to use $gtm_passwd to access the database. Note that GT.M would not permit access to the database with the hashed passphrase set in $gtm_passwd if $gtm_obfuscation_key is not available in the environment. FIS recommends setting the gtm_passwd environment variable using an $gtm_obfuscation_key. For more information on the gtm_passwd and gtm_obfuscation_key environment variables, refer to "Environment Variables" [19].

Remember that $gtm_passwd is not a database authenetication mechanism. $gtm_passwd provide the keyring passphrase to GT.M and the requirement to put it in an obfusacated form is for better security.

Master Key Configuration File and Encryption Keys

The reference implementation uses the database section of the $gtmcrypt_config file to obtain the symmetric keys for encrypting a database file. The environment variable gtmcrypt_config specifies the location of the master key configuration file which contains dat and key combinations. A dat entry specifies the absolute location of the database file and the key entry specifies the absolution location of the encryption key. The master key configuration file leverages the popular libconfig library (http://www.hyperrealm.com/libconfig). Please refer to "Why do we need a $gtmcrypt_config file?" (page 514) for instructions on creating the master key configuration file.

Note that encryption 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.

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:

$gtm_dbkeys

Master Key File
dat dbfile 1 key keyfile 1
dat dbfile 2 key keyfile 2

Key File
Symmetric key for dbfile 1 encrypted with Public key

Private Key*

Key File
Symmetric key for dbfile 2 encrypted with Public key

Memory Key Ring
Symmetric key for dbfile 1

Keyring on disk

$gtm_passwd

Memory Key Ring
Symmetric key for dbfile 1

Encrypted
Database
dbfile 1 +
Journal file

Encrypted
Database
dbfile 2 +
Journal file

* 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, MUPIP 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, MUPIP 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:

1. Helen and Phil each create a new GPG keyring and a new public-private key pair[3]. In the gen_keypair.sh script GPG generates the key pair[4], 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[5]. This is illustrated below; first Helen, then Phil (if the GNUPGHOME environment variable is not set, it will default to **$HOME/.gnupg**).
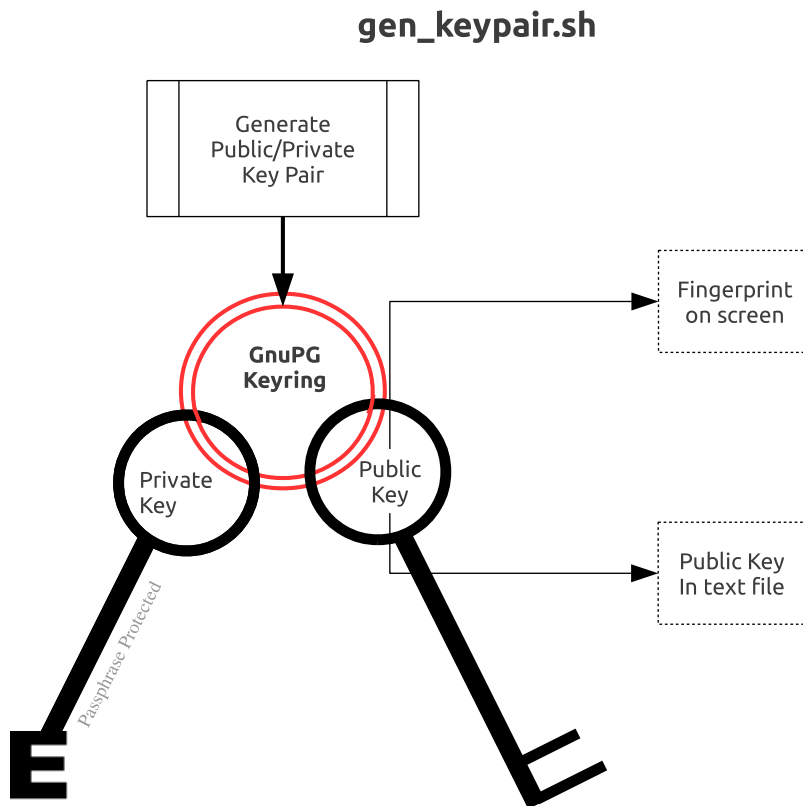
2. Helen e-mails helen@gt.m_pubkey.txt the file containing her exported public key to Phil, and Phil sends phil@gt.m_pubkey.txt, his exported public key to Helen. To protect against "man in the middle" attacks, they speak on the phone to exchange keyfingerprints, or send each other the fingerprints by 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
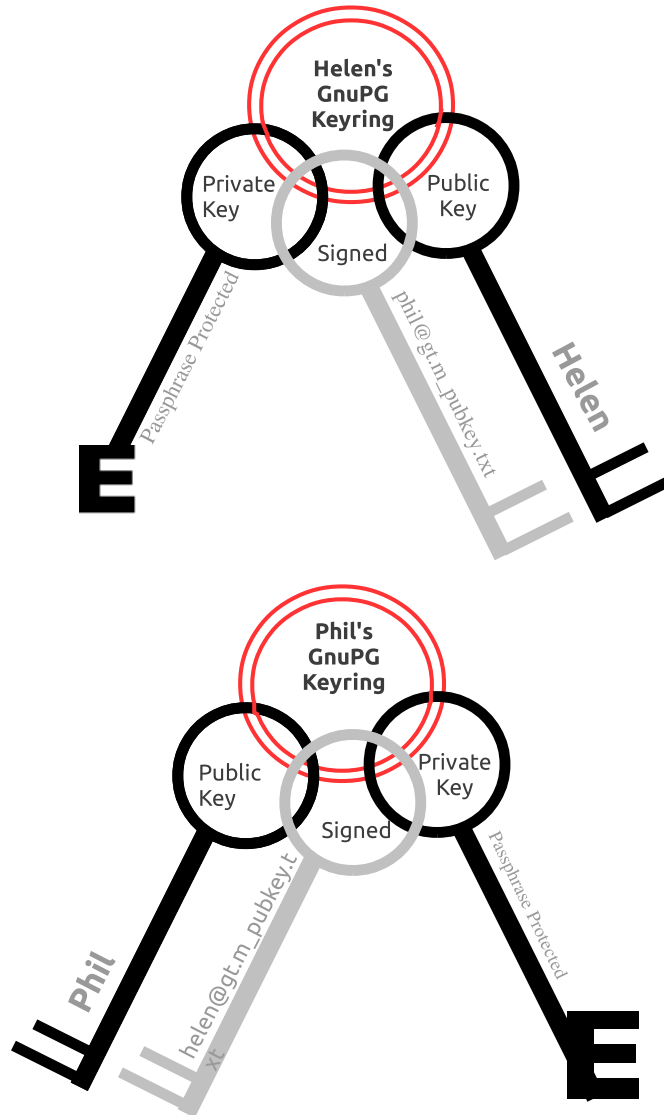
3. Using the gen_sym_key.sh script, Helen generates a symmetric cipher key for Phil to use in encrypting a new database file cust.dat. With a key strength of 2, a symmetric key is suitable for use in production and in the example is stored in file helen_cust_dat.txt encrypted with Helen's public key 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 Phi'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

Helen's
GnuPG
Keyring

Private
Key

Public
Key

Signed

Symmetric
Key

Passphrase Protected

phil@gt.m_pubkey.txt

Helen

helen_cust_dat.txt

encrypt_sign_db_key.sh

Signed
Symmetric
Key

phil_cust_dat.txt
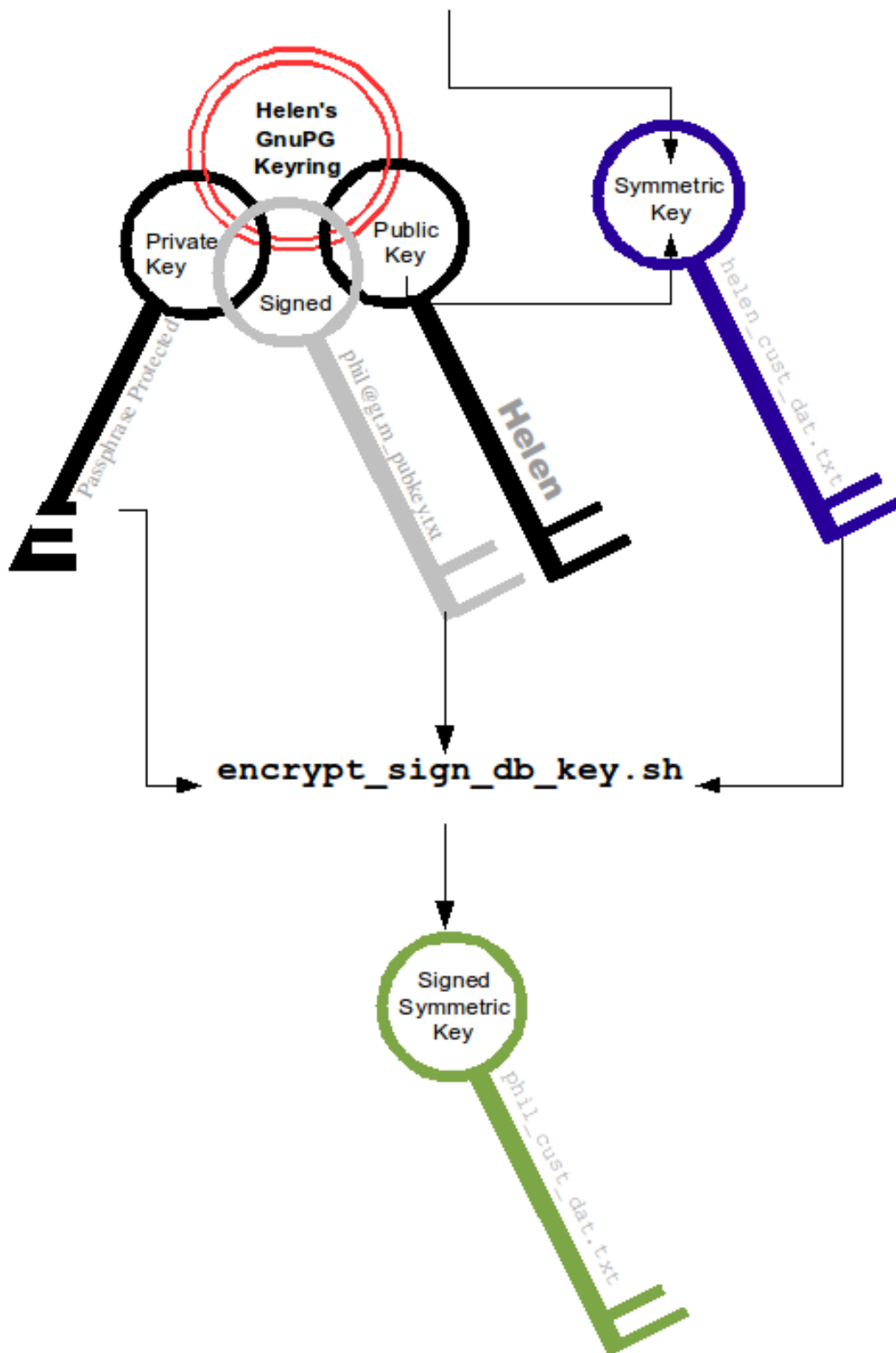
5. With the add_db_key.sh script, Phil now adds the key to his GT.M master key file. He can then create the encrypted database file with mupip create, load it with data and use it. Until the database is created and loaded with data, the key has no value and can be discarded at will. Once the database 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:
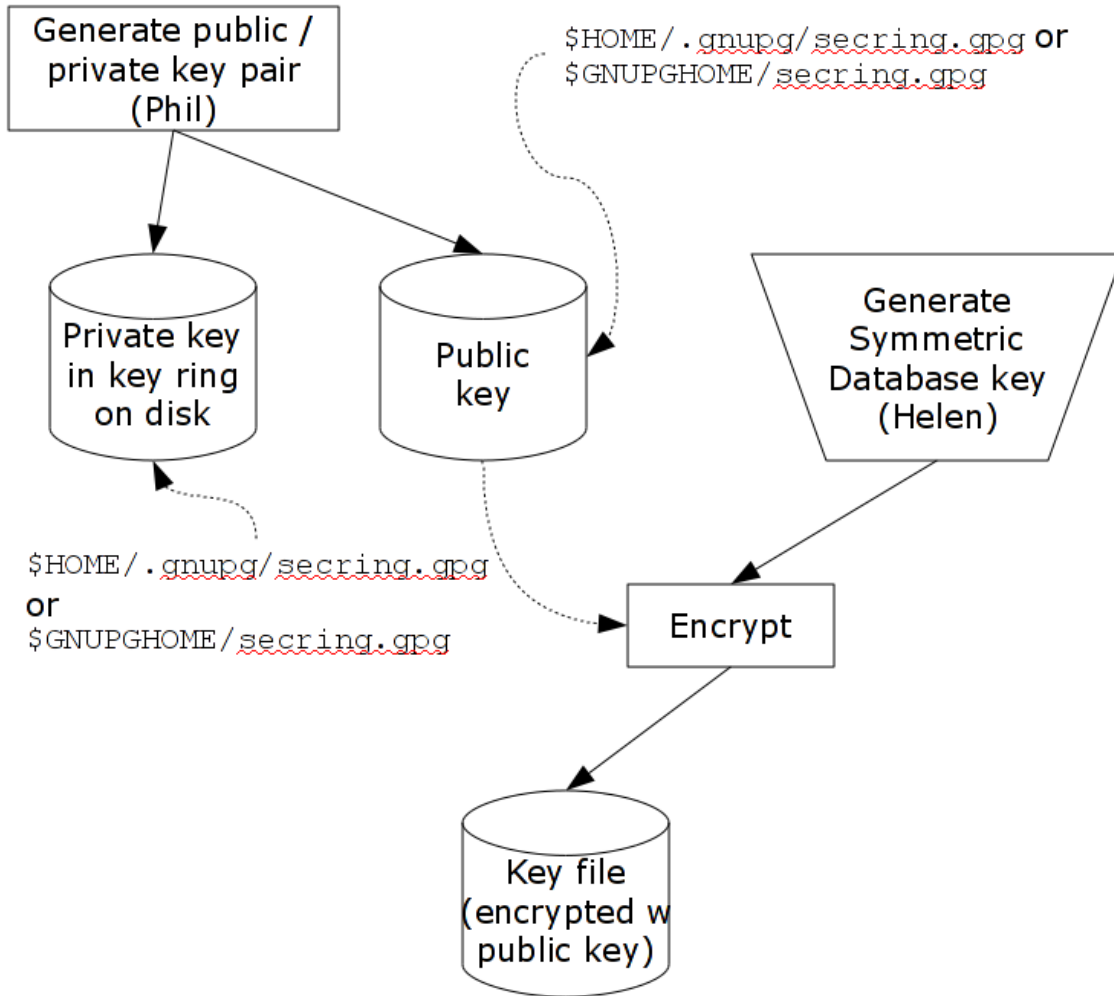
6. As a final check to make sure that the database was created with the correct symmetric cipher key and the correct cipher, Helen can use the gen_sym_hash.sh script to compute a hash from the key in helen_cust_dat.txt while Phil uses GT.M's dse dump -fileheader -all command to print the key from the 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:

```
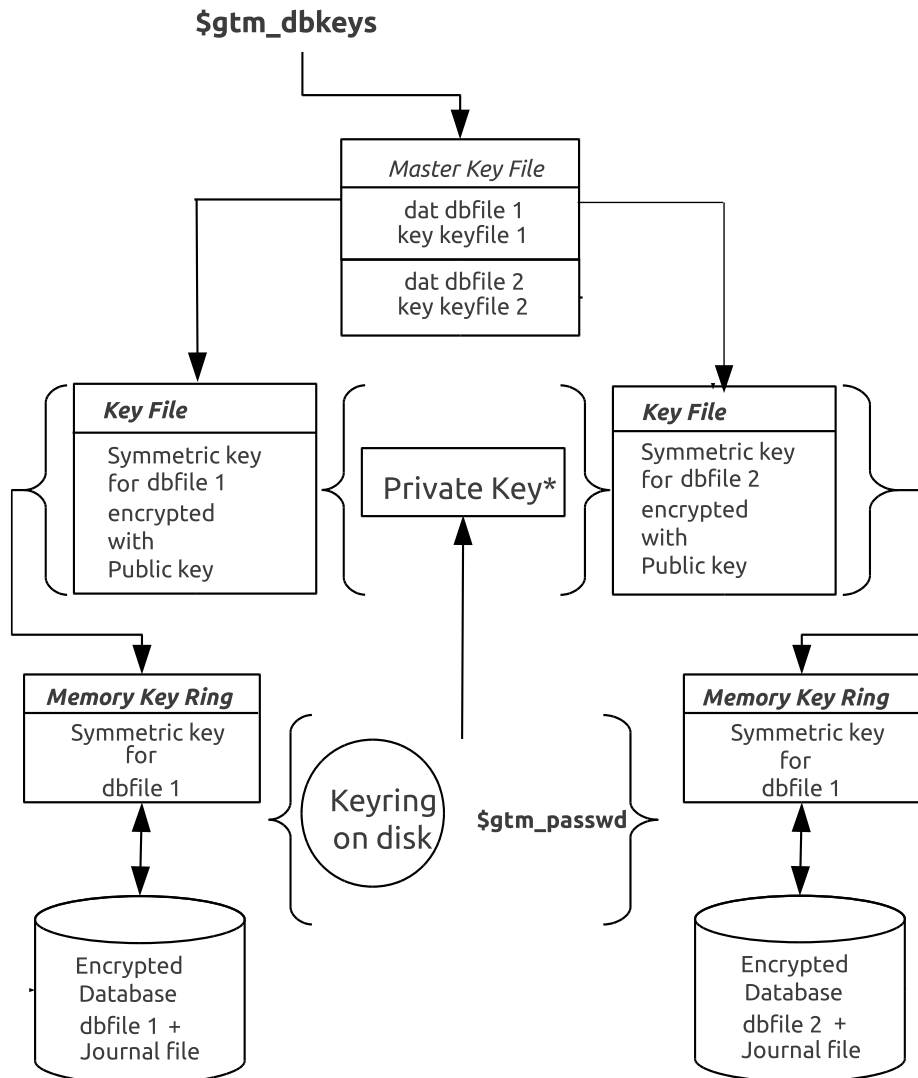helen$ export GNUPGHOME=$PWD/.helengnupg
helen$ $gtm_dist/plugin/gtmcrypt/gen_keypair.sh helen@gt.m Helen Keymaster
Passphrase for new keyring:
Verify passphrase:
Key ring will be created in /home/helen/.helengnupg
```

```
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
#######################################################
```

```
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 MUPIP 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 MUPIP 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
```

```
/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
 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$ $gtm_dist/plugin/gtmcrypt/gen_sym_hash.sh helen_cust_dat.txt
Passphrase for keyring:gpg: encrypted with 2048-bit RSA key, ID A2E8A8E8, created 2010-05-07"
Helen Keymaster <helen@gt.m>"178E55E32DAD6BFF761BF917412EF31904C...
helen$
```

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.
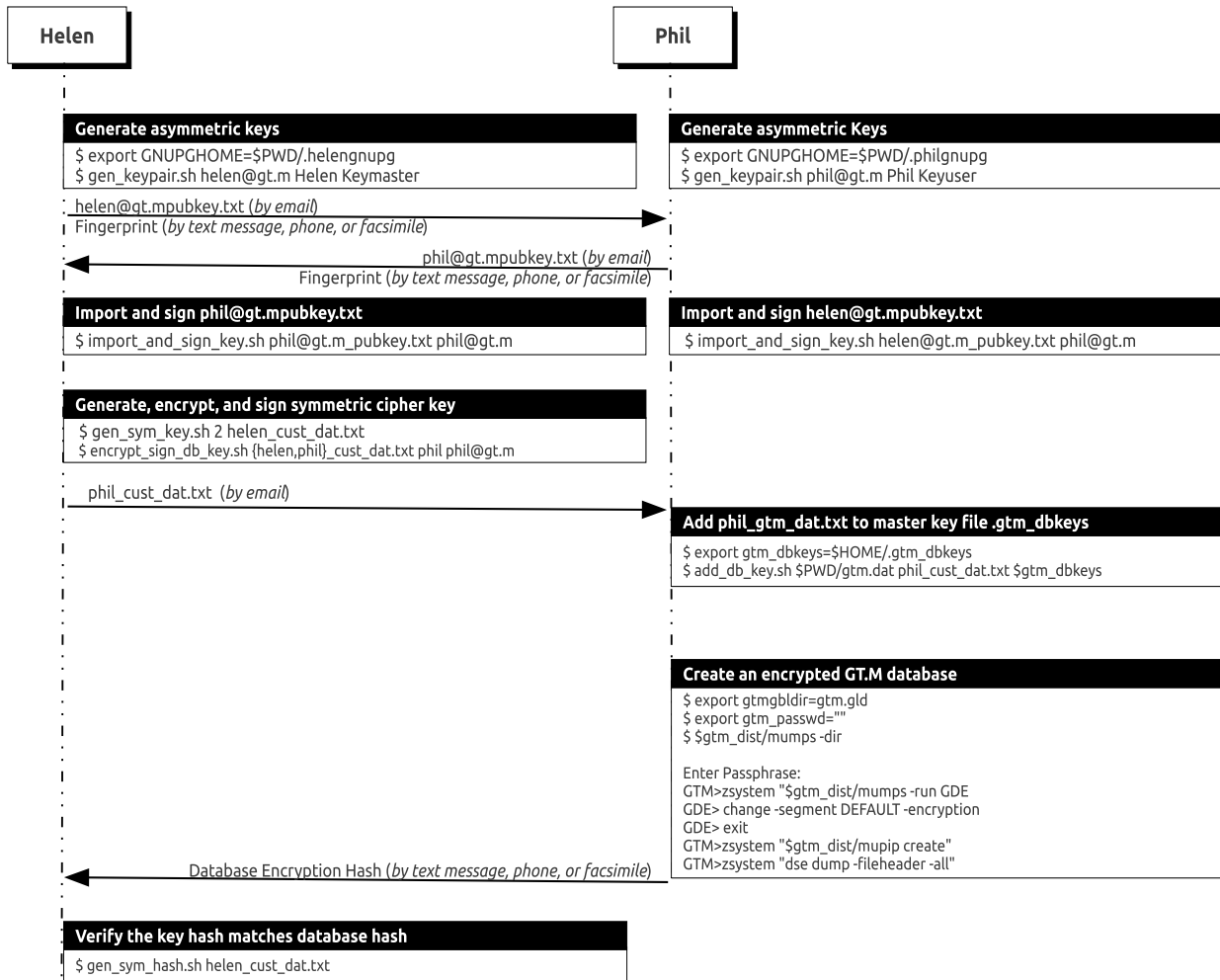
Tested Reference Implementations

GT.M database encryption comes with a source reference implementation that should compile "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 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 18.04 LTS	1.10.0	1.27-unknown	libgcrypt 1.8.1	2.2.4
Ubuntu 20.04 LTS	1.13.1-unknown	1.37	libgcrypt 1.8.5	2.2.19
Ubuntu 22.04 LTS	1.16.0-unknown	1.43	libgcrypt 1.9.4	2.2.27
RHEL 7	1.3.2	1.12	libgcrypt 1.5.3	2.0.22
RHEL 8	1.13.1	1.31	libgcrypt 1.8.5	2.2.20
AIX 7.1	1.4.0	1.11	libgcrypt 1.5.1 libcrypto from OpenSSL - (version >= 1.5) AES256CFB as implemented by OpenSSL - (version >= 0.9.8)	1.4.11

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 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, compiling the source reference implementation produces the shared library plugins, libgtmcrypt_gcrypt_AES256CFB.so and libgtmcrypt_openssl_AES256CFB.so. On installation, platforms other than AIX, libgtmcrypt.so is a symbolic link to libgtmcrypt_gcrypt_AES256CFB.so; on AIX symbolic link is to libgtmcrypt_openssl_AESCFB.so.

> **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 454) section.

While GT.M dropped support for Blowfish in V6.3-001 and FIS no longer tests it, you may continue to use Blowfish CFB from V6.0-001 through V6.3-000A using the following information. 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. In order to use Blowfish CFB after 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:

- 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. 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.

- If $gtm_crypt_plugin is not defined, GT.M expects $gtm_dist/plugin/libgtmcrypt.so to be a symbolic link to a shared library providing the plugin. The expected name of the actual shared library is libgtmcrypt_cryptlib_CIPHER.so (depending on your platform, the actual extension may differ from .so), for example, libgtmcrypt_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**
>
> GT.M V6.3-001 dropped support for the Blowfish encryption plugin.
>
> 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 230) for more complete documentation.

Special note - GNU Privacy Guard and Agents

The GNU Privacy Guard (GPG) supports the use of an agent to manage encrypted keys. Agents allow for protocol independent access to keys stored in users' GPG keyrings.

FIS strongly recommends using a separate keyring and configuration for GT.M applications. The GPG keyring and related configuration files reside in $GNUPGHOME. Using a separate $GNUPGHOME insulates the GT.M application from interference with any user desktop/workstation environment. Configuration options necessary to support GT.M could negatively impact other programs and vice versa.

Starting with GPG version 2, GPG required the use of the agent. However, in testing, FIS has found that GPG Classic versions 1.4.16 and up, may also require an agent. While the following information is valid as of GPG release 2.1.18, later versions may introduce some wrinkles in the agent operation. Users must familiarize themselves with GPG while setting up encryption.

While GPG comes with an agent program, gpg-agent, other parties often provide their own agent implementation, e.g. gnome-keyring-daemon. These third party agents often start up, as a convenience, during user login to provide password and key caching services. Agents typically define GPG_AGENT_INFO in the environment pointing to a socket file. Since third-party agents define GPG_AGENT_INFO in the environment, GT.M scripts must undefine it to avoid communicating with the third party agents. It is possible that these third-party agents create the GPG default socket file $GNUPGHOME/S.gpg-agent. Using a separate $GNUPGHOME insulates a GT.M application from third part agents.

When invoking GPG via GPGME, there is no convenient way to avoid invoking an agent 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. By default the GPG agent calls /usr/bin/pinentry the pinentry program. FIS provides a custom pinentry function for GT.M's encryption reference implementation (packaged in pinentry-gtm.sh and pinentry.m).

Spurious CRYPTKEYFETCHFAILED errors

A defect that affects GnuPG 2.0+ versions causes the gpg-agent to fail decrypting the GnuPG private key that secures the database encryption key. This decryption failure results in spurious CRYPTKEYFETCHFAILED errors during process startup or re-encryption. This defect appears more frequently with GnuPG releases starting at 2.1.15. At the time of this writing, Ubuntu 17.04 - 17.10, Debian 9 and Fedora 26 - 27 all have the affected GnuPG versions. However Fedora 26 - 27 are slated to receive fixed versions.

GPG versions 2.1.15 and up suffer from persistent CRYPTKEYFETCHFAILED errors. The only recommended course of action is to upgrade to GnuPG 2.2.4 and libgcrypt 1.8.2 which contain the fixes for the defects https://dev.gnupg.org/T3473 and https://dev.gnupg.org/T3530. The GPG fixes that address the CRYPTKEYFETCHFAILED errors require additional gpg-agent configuration options listed below. Please kill and restart any existing GPG agent processes for the agent to pick up the changes.

Using the reference implementation's custom pinentry program

pinentry-gtm.sh is a custom pinentry program that prevents prompting for keyring passphrase by Gnu Privacy Guard operations when the environment variable gtm_passwd is already defined. When there is a GETPIN request and the gtm_passwd environment variable is defined, pinentry-gtm.sh runs pinentry.m and returns the to the calling program. Custom pinentry programs like pinentry-gtm.sh are meaningful only when you set gtm_passwd to an obfuscated passphrase. When the

environment variable gtm_passwd is not defined or a usable mumps or pinentry.m does not exists, pinentry-gtm.sh runs the default pinentry program and prompts for passphrase. Remember that pinentry.m can reveal the passphrase. Therefore, ensure that you restrict the access for the pinentry.m's object file to only those users who manage your keys. GT.M provides pinentry-gtm.sh as a convenience to those users who are bothered by prompting for keyring passphrases for Gnu Privacy Guard related operations. Neither pinentry-gtm.sh nor pinentry.m is used internally by any GT.M database operation.

> ### Note
>
> When you set gtm_passwd to "", GT.M obtains the passphrase using the default GTMCRYPT passphrase prompt. When gtm_passwd is set to "", you can neither use a pinentry program (custom or default) to obtain a passphrase nor customize the default GTMCRYPT passphrase prompt.

To use the custom pinentry program, you need to perform the following setup actions:

1. At the OS level, ensure that the default pinentry program for servers is the "curses" pinentry executable and not the GUI version. Should the custom pinentry program fail, GPG invokes the default pinentry program. If the default pinentry program is for the GUI, a console user typically would not become aware of the password request.

   For Redhat systems use 'yum search pinentry' to search for the available pinentry programs for the "curses" version.

   For Debian and Ubuntu systems use 'apt search pinentry' to search for the available pinentry programs for the "curses" version.

2. GT.M scripts must undefine GPG_AGENT_INFO

3. GT.M scripts must define GPG_TTY or the (GPG 2.1 and up) pinentry program may not work. e.g.:

   ```
   export GPG_TTY=$tty
   ```

4. Set up the encryption keys using the gen_keypair.sh script. This script creates a file gpg-agent.conf in the GnuPG directory (specified by the environment variable $GNUPGHOME) with the following line directing GPG to invoke the reference implementation's custom pinentry program.

   ```
   pinentry-program <path to $gtm_dist>/plugin/gtmcrypt/pinetry-gtm.sh
   ```

   When pinetry-gtm.sh finds the environment variable $gtm_passwd defined and an executable GT.M, it runs the pinentry.m program which provides GnuPG with the keyring password from the obfuscated password. Otherwise, it calls /usr/bin/pinentry.

5. The custom pinentry program uses a GT.M external call. Each GT.M application that uses pinentry-gtm.sh must define the environment variable GTMXC_gpgagent to point to the location of gpgagent.tab. By default, the reference implementation places gpgagent.tab in the $gtm_dist/plugin/ directory. gpgagent.tab is an external call table that pinentry.m uses to unmask the obfusacated password stored in gtm_passwd.

6. Direct the gpg-agent to use it's standard Unix domain socket file, $GNUPGHOME/S.agent, when listening for password requests. Enabling the standard socket simplifies the gpg-agent configuration. Enable the standard socket by adding the following configuration option to $GNUPGHOME/gpg-agent.conf.

   ```
   echo "use-standard-socket" >> $GNUPGHOME/gpg-agent.conf
   ```

7. When using GPG 2.1.12 and up, enable loopback pinentry mode by adding the following configuration option to $GNUPGHOME/gpg-agent.conf. With this option in place, the agent can call back to GT.M directly for the passphrase if GPG directs it to do so.

```
echo "allow-loopback-pinentry" >> $GNUPGHOME/gpg-agent.conf
```

8. When using GPG 2.1.12 and up with GT.M versions prior to V6.3-001, you can bypass the agent by forcing GPG to use pinentry loopback mode, by adding the following configuration option to $GNUPGHOME/gpg.conf. This eliminates the custom pinentry progam configuration.

```
echo "pinentry-mode=loopback" >> $GNUPGHOME/gpg.conf
```

9. When using GPG 2.2.4 and up use the option to auto-increase secmem in gpg-agent (https://dev.gnupg.org/T3530)

```
echo "auto-expand-secmem" >> $GNUPGHOME/gpg-agent.conf
```

10. When using GPG 2.2.4 and up use the option to increase the configurable backlog for sockets (https://dev.gnupg.org/T3473)

```
echo "listen-backlog 128" >> $GNUPGHOME/gpg-agent.conf
```

For more information on other available options for your gpg-agent.conf, refer to the manpages of gpg-agent (man gpg-agent).

> ⚠️ **Warning**
>
> 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. Depending upon the gpg version ("man gpg" to confirm) you can override the agent configuration. Otherwise, you will need to temporarily comment out the pinentry-program line in gpg-agent.conf by placing a "#" in front of the line, e.g.:
>
> ```
> #pinentry-program <path to $gtm_dist>/plugin/gtmcrypt/pinetry-gtm.sh
> ```
>
> The encryption plugins included with GT.M releases prior to V5.4-001 are not compatible with GPG agents.

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) does not automatically install GT.M with the reference implementation plug-in. You need to follow the compilation instructions in the Plugin Architecture and Interface section

If the encryption libraries are not part of the automatic search path on your system, you 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 $gtmcrypt_config.

The structure of the $gtm_dist/plugin directory on Linux x86 after plugin compilation is as follows:

```
plugin/
|-- gpgagent.tab
|-- gtmcrypt
|   |-- Makefile
|   |-- README
|   |-- encrypt_sign_db_key.sh
|   |-- gen_keypair.sh
|   |-- gen_sym_hash.sh
```

```
|   |-- gen_sym_key.sh
|   |-- gtm_tls_impl.c
|   |-- gtm_tls_impl.h
|   |-- gtm_tls_interface.h
|   |-- gtmcrypt_dbk_ref.c
|   |-- gtmcrypt_dbk_ref.h
|   |-- gtmcrypt_interface.h
|   |-- gtmcrypt_pk_ref.c
|   |-- gtmcrypt_pk_ref.h
|   |-- gtmcrypt_ref.c
|   |-- gtmcrypt_ref.h
|   |-- gtmcrypt_sym_ref.c
|   |-- gtmcrypt_sym_ref.h
|   |-- gtmcrypt_util.c
|   |-- gtmcrypt_util.h
|   |-- import_and_sign_key.sh
|   |-- maskpass
|   |-- maskpass.c
|   |-- pinentry-gtm.sh
|   |-- pinentry.m
|   |-- show_install_config.sh
|   `-- source.tar
|-- libgtmcrypt_gcrypt_AES256CFB.so
|-- libgtmcrypt_openssl_AES256CFB.so
|-- libgtmcrypt.so -> ./libgtmcrypt_gcrypt_AES256CFB.so
|-- o
`-- r
```

Administration and Operation of Encrypted Databases

Utility programs written in M (such as %GO) run within GT.M processes and behave like any other code written in M. Encryption keys are required if the GT.M 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 uitlities do not need access to encryption keys to operate with encrypted databases.

MUPIP and DSE use the same plug-in architecture as GT.M 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.

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 the Encryption Keys

FIS recommends rotating (changing) the encryption key of the database for better security. The frequency of encryption key rotation depends on your security requirements and policies. MUPIP REORG -ENCRYPT provides the option to encrypt a database or rotate the keys of an already encrypted database. If you are using replication, you can encrypt the replicating secondary instance first to prevent your originating primary instance from any additional IO load that MUPIP REORG -ENCRYPT may add. 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. For more information, refer to "-Encrypt" (page 138).

Encrypted Database Creation

To create a new encrypted database, first use GDE to flag the database file for encryption (with the -ENCRYPTION segment qualifier). Then, create a $gtmcrypt_config file and work with your security team to obtain the relevant encryption keys. Then, use the maskpass utility to set the environment variable gtm_passwd to the obfuscated form of the passphrase keyring. Finally, execute MUPIP CREATE to create an encrypted database and MUPIP REORG -ENCRYPT=<encr_key> to encrypt the blocks of the database.

Once you encrypt a database, you cannot turn off encryption, it stays encrypted even if you specify MUPIP SET -NOENCRYPTABLE. If you wish any of the data in an encrypted database to be available unencrypted, you must extract the data and load it into a new database created to be unencypted and appropriately map the new database with a global directory. You can also use a MERGE command and multiple global directories to move data in either direction between encrypted and unencrypted database files.

Plugin Architecture & Interface

As noted in the Tested Reference Implementations, GT.M includes the source code to 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 as ready to compile 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 source code by default resides in $gtm_dist/plugin/gtmcrypt/source.tar.

The reference implementation includes:

1. A $gtm_dist/plugin/gtmcrypt/source.tar archive with all source files and scripts. The archive includes a Makefile to build/ install the plugins and "helper" scripts, for example, add_db_key.sh. A brief description of these scripts is as follows:

show_install_config.sh	Reports the cryptographic library and cipher that a GT.M process would use, from $gtm_crypt_plugin, if it has a value and otherwise from the name of the library linked to by libgtmcrypt.so.
gen_sym_hash.sh	Uses show_install_config.sh to identify the currently installed encryption configuration so that it can generate the appropriate cryptographic hash for the provided symmetric key.
import_and_sign_key.sh	Imports and signs one another's public keys.
gen_sym_key.sh	Generates a symmetric cipher key for others to use in encrypting a database file.
encrypt_sign_db_key.sh	Uses a private key to decrypt the symmetric cipher key , encrypts it with other's public key, and signs it with the private key.
add_db_key.sh	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. A Makefile to build and install each of the encryption plugin libraries. The Makefile conforms to the regular use pattern of "make && make install && make clean". Building the reference plugin libraries[6] requires a compiler and development libraries for GPG and OpenSSL. The reference plugins are:

gpgagent.tab	Call-out interface table to let MUMPS programs unobfuscate $gtm_passwd
libgtmcrypt.so	A symlink to the default encryption library
libgtmcrypt_gcrypt_AES256CFB.so	The reference plugin that leverages GPG for encryption using the AES256CFB algorithm
libgtmcrypt_openssl_AES256CFB	The reference plugin that leverages OpenSSL for encryption using the AES256CFB algorithm
libgtmcryptutil.so	A reference plugin support library

libgtmtls.so	The reference plugin that leverages OpenSSL for transport encryption features for the MUMPS language
gtmpcrypt/maskpass	Program to mask the password stored in $gtm_passwd

Extensions to the GT.M External Interface

GT.M provides additional C structure types (in the gtmxc_types.h file):

1. gtmcrypt_key_t - a datatype that is a handle to a key. The GT.M database engine itself does not manipulate keys. The plug-in 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 libgpgme / libgcrypt.

The plugin interface functions are:

1. gtmcrypt_init()

2. gtmcrypt_getkey_by_name()

3. gtmcrypt_getkey_by_hash()

4. gtmcrypt_hash_gen()

5. gtmcrypt_encrypt()

6. gtmcrypt_decrypt()

7. gtmcrypt_close()

8. and gtmcrypt_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 gtmcrypt* functions are capable of managing multiple keys for multiple database files. Prototypes for these functions are in gtmcrypt_interface.h.

The core plugin interface functions, all of which return a value of type xc_status_t are:

- gtmcrypt_init() performs initialization. If the environment variable $gtm_passwd exists and has an empty string value, GT.M calls gtmcrypt_init() before the first M program is loaded; otherwise it calls gtmcrypt_init() when it attempts the first operation on an encrypted database file.

- Generally, gtmcrypt_getkey_by_hash or, for MUPIP CREATE, gtmcrypt_getkey_by_name perform key acquisition, and place the keys where gtmcrypt_decrypt() and gtmcrypt_encrypt() can find them when they are called.

- Whenever GT.M needs to decode a block of bytes, it calls gtmcrypt_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 gtmcrypt_encrypt() to encrypt the data.

- If encryption has been used (if gtmcrypt_init() was previously called and returned success), GT.M calls gtmcrypt_close() at process exit and before generating a core file. gtmcrypt_close() must erase keys in memory to ensure that no cleartext keys are visible in the core file.

More detailed descriptions follow.

- gtmcrypt_key_t *gtmcrypt_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 gtmcrypt_hash_gen(gtmcrypt_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, gtmcrypt_hash_gen() must fill any unused space in the 256 byte buffer with zeros.

- gtmcrypt_key_t *gtmcrypt_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 gtmcrypt_encrypt(gtmcrypt_key_t *key, xc_string_t *inbuf, xc_string_t *outbuf) and xc_status_t gtmcrypt_decrypt(gtmcrypt_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), gtmcrypt_decrypt() must be able to decrypt any data buffer encrypted by gtmcrypt_encrypt(), otherwise the encrypted data is rendered unrecoverable[7]. As discussed earlier, GT.M requires the encrypted and cleartext versions of a string to have the same length.

- char *gtmcrypt_strerror() - GT.M uses this function to retrieve addtional 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 Reference Implementation with Older Releases

The interface between GT.M and the encryption libraries has changed over time. Using a reference implementation different from the version that shipped with GT.M is not recommended. Custom encryption libraries should be verified to work with newer version before deployment into production.

Chapter 13. GT.CM Client/Server

Revision History		
Revision V7.0-004	20 September 2022	• In "OpenSSL Options" (page 516), remove dh params • In "Why do we need a $gtmcrypt_config file?" (page 514), Eliminate dh512 and dh1024 • In "Creating and signing demo leaf-level certificates" (page 519), no dh params
Revision V6.3-014	06 October 2020	• In "Why do we need a $gtmcrypt_config file?" (page 514), tlsids must begin with a letter.
Revision V6.3-004	23 March 2018	• In "Recommendations" (page 498), add recommendation for using restrictions to limit what facilities are available to GT.M applications • In "Philosophy" (page 496), add safety checks for system()
Revision V6.3-003	12 December 2017	• In "Building Encryption Libraries " (page 495), AIX pre-reqs
Revision V6.3-002	22 August 2017	• In "Building Encryption Libraries " (page 495), eliminate old platforms and reduce instruction bloat for linuxen • In "IBM AIX 7.1 (pSeries) " (page 495), libgcrypt is an AIX requirement • In "Overview" (page 504), add information about the gtmpcat_for_{release} kit. • In "Usage" (page 505), add callin, cmdscript, gtmdeftypes, ppi, and regiondetail options.
Revision V6.3-001	20 March 2017	• In "Monitoring GT.M Messages" (page 490), improved the note on gtm_procstuckexec • In "GT.CM Client" (page 480), added clarity to the GT.CM client requirements • In "Introduction" (page 479), removed references to OpenVMS and move the cross-endian statement from the server section to this section. • In "Recommendations" (page 498), removed references to Solaris. • In "gtmsecshr Exception" (page 497), removed references to Solaris and HP-UX.

		• In "Managing core dumps " (page 493), removed references to HP-UX and Solaris.
		• In "Overview" (page 504), removed references to HP-UX and Solaris.

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.

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.

GT.CM can communicate between systems having different endian architectures.

Notes

- 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.

- 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.

The Global Directory of the client specifies a GT.CM database segment by prefixing its file with an alphanumeric <node-name>, followed by a colon (":"). Client processes using this database must have an environment variable of the form "GTCM_<node-name>" to locate the server. This environment variable may contain either a port number alone, or a host name or address and a port number in the form "<host-name-or-address>:<port-number>" or the form "[<host-name-or-address>]:<port-number>", where the square-brackets ([])are part of the text. If the port number is specified alone, GT.CM uses the <node-name> as the host name; otherwise, it uses the <node-name> solely as an identifier to match the segment in the Global Directory, and it obtains the host name or address from the contents of the environment variable. If a host name is specified, and the server host has multiple addresses, GT.CM uses the system default.

> ## Notes
>
> - Because the <node-name> is strictly alphanumeric, it cannot represent an IP address or qualified host name due to the need for a dot (".") separator. If you need to specify an IP address or qualified host name, select an alphanumeric <node-name> and specify the <host-name-or-address> in the GTCM_<node-name> variable.
>
> - Use the "[<host-name-or-address>]:<port-number>" form to specify an IPv6 address.

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
```

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/ gtcm_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

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()**.

- 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.

6. 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 [0x00260002] :: 0xf53803000000000000fe000000000000ffffffd2
```

7. 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] :: 0xf73803000000000000fe000000000000ffffffd2
```

Note that the **Semaphore id 1015810** and **Shared Memory ID 2457601** are in the sample output of the **ipcs -a** command below.

8. 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
0x00000000 1081348 welsley 777 3

------ Message Queues --------
key  msqid owner perms used-bytes messages
```

Using the following formula, where n is the number of regions, to calculate GT.M's IPC resources in a multisite replication configuration:

```
IPCs = (n regions * (1 shm/region + 1 ftok sem/region + 1 private sem/region))
+ 1 sem/journal-pool + 1 sem/receiver-pool
```

In this case, America has one region and no receiver-pool so:

```
1 region * 3 IPCs/region + 1 IPC/journal-pool = 4 IPCs
```

Therefore, assuming that instance **America** has 1 region, the total IPC utilized by GT.M is: 4 [1 * 3 + 1 +0]. Note that there is no receiver pool for instance **America**.

> **Note**
>
> For **MUPIP RECOVER** operations the total number of IPC resources are **3n** (As there is no Journal Pool or Receiver Pool) where **n** is the number of regions.

9. Now connect to **Server_B** and give the following commands in the directory holding database files for **Brazil**:

```
$ export gtm_repl_instance=multisite1.repl
$ mupip replicate -instance_create -name=Brazil $ mupip rundown -region "*"
$ mupip set -journal="enable,before,on" -replication=on -region "*"
$ mupip replicate -source -start -passive -buf=1048576 -log=B2dummy.log -inst=dummy
$ mupip replicate -receive -start -listenport=1234 -buf=1048576 -log=BFrA.log
```

Now execute the command:

```
$gtm_dist/ftok mumps.dat multisite1.repl
```

This command produces the "public" (system generated) IPC Key of **mumps.dat** and its replication instance **multisite1.repl**. It produces a sample output like the following:

```
  mumps.dat :: 722134735 [ 0x2b0ae6cf ]
multisite1.repl :: 722134737 [ 0x2b0ae6d1 ]
```

Note that keys starting with 0x2b in the output of the ipcs -a command are the public IPC keys for the semaphores of the database file on replication instance **Brazil**.

10. Then, execute the following command and note down the **shared memory id** and **private semaphore id** on instance **Brazil**.

```
$ 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 :: 327683 [0x00050003] :: 11665410 [0x00b20002]:: 0xcfe6340000000000a000000000000000000000
```

11. Now, execute the following command and note down the **shared memory** and **private semaphore id** for journal pool.

```
$ 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]:: 0xd1e6340000000000a000000000000000000000
```

Note that the **Semaphore id 262145** and **Shared Memory ID 11632641** are in the sample output of the **ipcs -a** command below.

---

12.   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 **gtmsecshr** as owned by root and with the **setuid bit on**. **gtmsecshr** 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. **gtmsecshr** is guarded by a wrapper program. The wrapper program protects **gtmsecshr** in the following ways:

* It restricts access to **gtmsecshr** in such a way that processes that do not operate as root cannot access it except though the mechanism used by the wrapper.

* Environment variables are user-controlled input to **gtmsecshr** and setting them inappropriately can affect system operation and cause security vulnerabilities. While **gtmsecshr** 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.

* gtmsecshr 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 **gtmsecshr** through socket files in a directory specified by the environment variable gtm_tmp.

gtmsecshr 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 **gtmsecshr** must be explicitly shut down are when a GT.M version is being removed - either a directory containing the GT.M version the running **gtmsecshr** 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 **gtmsecshr** 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. **gtmsecshr** 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

Revision History		
Revision V6.2-002	17 June 2015	Changed all occurences of GTM_FATAL_ERROR* to GTM_FATAL_ERROR*.txt.
Revision V6.2-001	27 February 2015	Added a new section called "Managing core dumps " (page 493).
Revision V6.0-000/1	21 November 2012	Added a table of replication message identifiers that are logged in the operator log.

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 GT.M 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

Component	Instance File or Replication Journal Pool	Receiver Pool	Identifier
	N	N/A	MUPIP
Receiver Server	Y	N/A	RCVSRVR
	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 309) 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 complete 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_*.txt** 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. For example for the script to invoke the debugger, the process must be of the same group or have a way to elevate privileges.

Managing core dumps

When an out-of-design situation or a fatal error causes a GT.M process to terminate abnormally, GT.M attempts to create a GTM_FATAL_ERROR.ZSHOW_DMP_*.txt file containing a dump of the M execution context. On encountering an unexpected process termination, GT.M instructs the operating system to generate a core dump on its behalf at the location determined from the core generation settings of the operating system. GTM_FATAL_ERROR*.txt and core dump files may help GT.M developers diagnose and debug the condition which resulted in an unexpected process termination, and help you get back up and running quickly from an application disruption. In additional to information having diagnostic value, a core dump file may also contain non-public information (NPI) such as passwords, local variables and global variables that may hold sensitive customer data, and so on. If you are an organization dealing with non-public information, you should take additional care about managing and sharing GTM_FATAL_ERROR.ZSHOW_DMP_*.txt and core dump files.

As core dump files may contain non-public information, you might choose to disable core dump generation. In the absence of a core dump file, you may be asked to provide detailed information about your hardware, GT.M version, application state, system state, and possibly a reproducible scenario of the unexpected process termination. Note that unexpected process terminations are not always reproducible. You are likely to spend a lot more time in providing post-mortem information during a GT.M support engagement than what you'd spend when a core dump file is available.

Core file generation and configuration are functions of your operating system. Ensure that core file generation is configured and enabled on your operating system. On Linux platforms, /proc/sys/kernel/core_pattern determines the naming convention of core files and /proc/sys/kernel/core_uses_PID determines whether the process id of the dumped process should added to the core dump file name. A core_pattern value of core creates core dump files in the current directory. Check the man mage for core (on Linux), and chcore (on AIX) for instructions on enabling and configuring core dump file generation according to your requirements.

> **Note**
>
> As maintainers of FIS GT.M, our goal is to make the product as reliable as it can be, so you should get few if any core files. Before a public release, GT.M goes through several rounds of automated testing which provides a thorough test coverage for new functionality and possible regressions. When new functionality passes our regression testing cycle, we
>
> frequently
> make field test releases so that GTM gets additional testing coverage in customer environments before a public release. While prioritizing fixes for a GT.M public release, we assign a higher priority to unexpected process terminations that our regression testing cycle and supported GT.M customers may report. As part of any fix, we add new test cases that become an integral part of future regression testing cycles. We have followed this practice for the past several years and therefore it is very unusual for a stable production application to generate core files. GT.M supplies a wide range of functionality in ways intended to maximize performance. Nonetheless, GT.M is reasonably complex as the number of possible execution paths is large, and our testing coverage may not include all possible edge cases. If you encounter a core dump because of a GT.M issue, it is likely that it is not part of our test coverage and we may find it hard to reproduce. Core files, especially combined with gtmpcat, are a powerful tool in diagnosing and addressing issues that cause process

failures. Note also that user actions can directly cause core files without any contributing GT.M issue (see the following example).

The following suggestions may help with configuring core dump files:

- Always put cores in a directory having adequate protection and away from normal processing. For example, the core file directory may have write-only permissions for protection for almost all users.

- Set up procedures to remove core dumps and GTM_FATAL_ERROR.ZSHOW_DMP_*.txt when they are no longer needed.

- Always configure core file generation in a way that each core gets a distinct name so that new cores do not overwrite old cores. GT.M never overwrites an existing core file even when /proc/sys/kernel/core_uses_pid is set to 0 and /proc/sys/kernel/core_pattern is set to core. If there is a file named core in the target core directory, GT.M renames it to core1 and creates a new core dump file called core. Likewise, if core(n) already exists, GT.M renames the existing core to core(n+1) and creates a new core dump file called core.

- Here are the possible steps to check core file generation on Ubuntu_x86 running GT.M V6.1-001_x86_64:

```
$ ulimit -c unlimited
$ /usr/lib/fis-gtm/V6.1-001_x86_64/gtm
GTM>zsystem "kill -SIGSEGV "_$j
$GTM-F-KILLBYSIGUINFO, GT.M process 24570 has been killed by a signal 11 from process 24572 with userid number
 1000
$ ls -l core*
-rw------- 1 gtmnode jdoe 3506176 Aug 18 14:59 core.24573
```

In order to test your core generation environment, you can also generate a core dump at the GT.M prompt with a ZMESSAGE 150377788 command.

- If you do not find the expected dump files and have already enabled core generation on your operating system, check file permissions and quotas settings.

- As GT.M core dumps are not configured for use with automated crash reporting systems such as apport, you might want to adjust the core naming conventions settings in such a way core dumps are preserved safely till the time you engage your GT.M support channel.

Before sharing a core dump file with anyone, you must determine whether the files contain NPI and whether the recipient is permitted to view the information in the files. GT.M Support at FIS does not accept NPI. You can use the gtmpcat software under the guidance of FIS GT.M Support to extract meaningful information from core files (by default, gtmpcat extracts meta-data without protected information; although you should always review any gtmpcat output before you send it to FIS). gtmpcat is diagnostic tool available to all customers who have purchased FIS GT.M Support. For more information on the gtmpcat software, refer to Appendix E: "GTMPCAT - GT.M Process/Core Analysis Tool " (page 504).

Appendix C. Building Encryption Libraries

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.

Debian, Ubuntu, RedHat and Fedora

Packages were installed from standard repositories using the package manager.

IBM AIX 7.1 (pSeries)

Dependencies: Building the encryption libraries on AIX requires GNU Make and IBM's xlc compiler toolchain.

GPG-ERROR

./configure CC=cc CFLAGS=-q64 ($OBJECT_MODE=64)

GCRYPT

./configure CC="xlc -q64" --disable-asm ($OBJECT_MODE=64)/ or CC=cc CFLAGS=-q64

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
```

Appendix D. GT.M Security Philosophy

Revision History		
Revision V6.2-001	27 February 2015	In "Shared Resource Authorization Permissions" (page 500), added factors that determine relink control file, relink control shared memory, and object shared memory permissions.

Philosophy

The general GT.M philosophy is to use the security of the underlying operating system, and to neither subvert it nor extend it. When GT.M invokes the POSIX system() interace, it ensures that alias and path settings do not interfere with the intentions of the actions it takes. The purpose of this document is to discuss the implications and limitations of this philosophy (the "security model"). This appendix reflects GT.M as of V6.3-004.

> **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 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.[1]

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

---

[1]The concept of write-only access to a database file is not meaningful for GT.M

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 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 GT.M process contains non-M code. If a GT.M 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 GT.M 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 499) .

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).

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).

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. Post installation, a system administrator can optionally add a restrict.txt file in $gtm_dist to restrict the use of certain GT.M facilities to a group-name. The owner and group for $gtm_dist/restrict.txt can be different from those used to install GT.M. For more information, refer to "Configuring the Restriction facility" (page 38)

4. 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.[2]. 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**

5. 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.

6. 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 **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.**

7. 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 are not considered trustworthy but which need read-only access to database files.[3]

8. 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.

9. 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.

10. 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.

11. Consider implementing layered security software if it exists for your platform, for example, SELinux, Trusted AIX.

> **Note**
>
> FIS neither endorses nor has tested any specific layered security product.

gtmsecshr commands

Commands	Action	Comments
WAKE_MESSAGE	Sends **SIGALRM** to specified process.	Used to inform receiving process that a resource (such as a critical section) it awaits has become available.
CONTINUE_PROCESS	Sends **SIGCONT** to specified process.	Used to awake a process that has been suspended while holding a resource. [a]
CHECK_PROCESS_ALIVE	Test sending a signal to specified process. [b]	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.
REMOVE_SEM	Remove a specified POSIX semaphore.	Used to remove an abandoned semaphore (for example, if the last attached process terminated abnormally).

Commands	Action	Comments
REMOVE_SHMMEM	Remove a specified shared memory segment.	Used to remove an abandoned shared memory segment. Before removing the segment, gtmsecshr checks that there are no processes attached to it.
REMOVE_FILE	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 (**msems**); unused on other platforms. Before removal, gtmsecshr verifies that the file is a socket file, in directory **$gtm_tmp**, and its name matches GT.M socket file naming conventions.
FLUSH_DB_IPCS_INFO	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.

[a]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**.
[b]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 takes a number of factors into account to determine the resulting permissions:

- The owner/group/other permissions of the database file or object directory

- The owner of the database file or object directory

- The group of the database file or object directory

- The group memberships of the database file's or object directory'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?	Execution of GT.M restricted to members of a group?
Group of Resource		*IPC Permissions***		*File Permissions****
-r--r--rw-	N	Y	N	N
Current group of process		*-rw-rw-rw-*		*-rw-rw-rw-*
-*--rw----	N	Y	Y	-
Group of database file		*-rw-rw----*		*-rw-rw----*
-r*-r*-r*-	-	-	Y	-
Group of database file		*-rw-rw-rw*		*-r*-r*-r**
-rw-rw-r*	-	-	N	-

For Autorelink permissions:

__*__ **: Routine directory**

__**__ **: rtnobj shared memory and relinkctl shared memory permissions. Note that rtnobj shared memory permissions have the x bit set wherever r or w are set.**

__***__ **: relinkctl file permissions**

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?	Execution of GT.M restricted to members of a group?
Group of Resource	*IPC Permissions** *			*File Permissions*** *
Current group of process	*-rw-rw-rw*			*-rw-rw-rw*
-rw-rw-rw	-	-	N	-
Current group of process	*-rw-rw-rw*			*-rw-rw-rw*
-rw-rw-rw	Y	Y	-	-
Group of database file	*-rw-rw-rw*			*-r*-r*----*
-r*-r*----	Y	N	-	N
Current group of process	*-rw-rw-rw-*			*-rw-rw-rw-*
-r*-r*----	Y	N	-	Y
Group to which GT.M is restricted	*-rw-rw----*			*-rw-rw----*
-r*-r*----	-	Y	-	-
Group of database file	*-rw-rw----*			*-r*-r*----*
-r*-r*----	-	N	-	N
Group of database file	*-rw-rw-rw-*			*-rw-rw-rw-*
-r*-r*----	-	N	-	Y
Group to which GT.M is restricted	*-rw-rw----*			*-rw-rw----*
----r*----	-	N	-	-
Group of database file	*-rw-rw----*			*----r*----*
-r*-------	Y	-	-	-
Current group of process	*-rw-------*			*-rw-------*

For Autorelink permissions:

* : Routine directory

** : rtnobj shared memory and relinkctl shared memory permissions. Note that rtnobj shared memory permissions have the x bit set wherever r or w are set.

*** : relinkctl file permissions

- 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.

- "Execution of GT.M restricted to members of a group" may remove "other" permissions.

Appendix E. GTMPCAT - GT.M Process/Core Analysis Tool

Revision History		
Revision V6.0-001/2	10 April 2013	In "Usage" (page 505), 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.

Starting with GT.M V6.3-002, each GT.M release includes a gtmpcat_for_{release} kit which contains the files required to run gtmpcat for the released GT.M version. install_gtmpcat.sh script installs the gtmpcat files in $gtm_dist/tools/gtmpcat. After installation, use $gtm_dist/gtmpcat to run gtmpcat. Note that the gtmpcat_for_{release} kit does not contain the files necessary for analyzing other GT.M releases. Full gtmpcat releases contain these files for all supported production GT.M releases as of the gtmpcat release date.

> ⚠️ **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) or dbx on AIX). 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.
--callin *<path>*		n/a	Specifies the path of the executable that created the core or the path to load in the given process to run against. Typically, this is a call-in type module that can also be used to diagnose LKE or DSE cores or processes.
--cmdscript *<entryref>*		-s	Use this option for the prototyping of additional information extraction. This option executes once every time you run GTMPCAT. This option also implies interactive mode. As soon as interactive mode setup is complete, GTMPCAT invokes the specified entryref. This options is like a gtmpcat-plugin in a way that it runs with full access to GTMPCAT routines, variables, and the debugger.
--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.
--gtmdeftypes		-g	Specifies the location of GTMDefinedTypesInit.m when it is not there in the current path.
--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 GT.M 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 .

GTMPCAT - GT.M Process/Core Analysis Tool

Options		Abbr	Description
--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.
--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 $gtmdbglvl 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.
--ppi	*	n/a	Specifies whether to extract personally identifying information from the core. If --ppi is not specified, GTMPCAT does not: • invoke the -l option. • extract local var information on the M stack. • dump database clues in the region output.
--regiondetail		n/a	Collects spanning regions information. Currently, this option collects data but does not display the output.
--tracedump		-t	Read and format the internal GTM trace table. Default is --notracedump. This is useful only to debug GT.M itself.

Options		Abbr	Description
--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>On<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 typdef structures defined in the GT.M header files and whose layouts are defined in GTMDefinedTypesInit.m[1].

- *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 F. 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 19). 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 additional 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 509), 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 510): 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 510) 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 GT.M 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 $ZTRAP="B"
export gtm_etrap='I 0=$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.

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 calcprint^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 calcprint^monthstarting() */
if ( 0 == gtm_init() ) gtm_ci("calcprint", &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("calcprint", &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 509).

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 calcprint() to the GT.M entryref **calcprint^monthstarting()**:

```
calcprint : gtm_int_t* calcprint^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" [510], 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_etrap environment** variable, as illustrated "Sample .profile" (page 510). 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_etrap**.

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 510) 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 GT.M 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).

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).

Appendix G. Creating a $gtmcrypt_config file

Revision History		
Revision V6.3-000A	20 March 2017	First published version.

Why do we need a $gtmcrypt_config file?

GT.M ships with a reference implementation of an encryption plugin which uses OpenSSL to perform various operations for Database Encryption, TLS replication and/or TLS-enabled sockets. The environment variable $gtmcrypt_config points to the configuration file that GT.M should use to set certain OpenSSL options. This configuration file must be in the structure of the libconfig configuration file. For more information on the structure of the libconfig configuration file, refer to http://hyperrealm.github.io/libconfig/libconfig_manual.html#Configuration-Files.

GT.M Database Encryption, TLS replication, and TLS sockets are not prerequisites for each other. You can use these GT.M features together or selectively in your environment depending on the $gtmcrypt_config file for a given process. The $gtmcrypt_config file may contain 2 groups -database and tls. Database Encryption uses the database group of the $gtmcrypt_config file. The database group should contain a list of database files and their corresponding key files. TLS replication and TLS sockets use the tls group of the $gtmcrypt_config file. The tls group contains information such as the location of the root certificate authority, leaf-level certificates with the corresponding private key files, and other TLS configuration options. Under the tls group, you can use the same tlsid or separate tlsids (with a different set of certificates and options) for TLS replication and TLS sockets depending on what establishes better security for your application. In your $gtmcrypt_config file, you do not need to add a database group if you are not using Database Encryption or a tls group if you are not using either TLS replication or TLS sockets.

Here is a sample $gtmcrypt_config file:

```
/* Database group */

database: {
      keys: (
          {
                  dat: "/tmp/mumps.dat";  /* Encrypted database file. */
                  key: "/tmp/mumps.key";  /* Encrypted symmetric key. */
          },
          {
                  dat: "/tmp/a.dat";
                  key: "/tmp/a.key";
          },
          ...
      );
}

/* TLS group */

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/";

      /* 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;

      /* ssl-options: specifies OpenSSL options to be set or cleared. */
      ssl-options: "SSL_OP_NO_SSLv2:SSL_OP_NO_SSLv3";

      /* List of alphanumeric TLS configuration identifiers (the "tlsid" used by GT.M processes) which must begin
with
a letter.
        * Each configuration consists of name/value pairs for specific options like certificate, key and format.
        */
      PRODUCTION: {
            /* Format of the certificate and private key pair. Currently, the GT.M TLS plug-in only supports PEM
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 gtmtls_passwd_<identifier>.
▶ For instance,
            * for the below key, the environment variable should be 'gtmtls_passwd_PRODUCTION'.
            * Currently, the GT.M TLS plug-in only supports RSA private keys.
            */
        key: "/home/jdoe/current/tls/certs/Malvern.key";
        /* Global options can be overridden here */
    };

    DEVELOPMENT: {
        format: "PEM";
        cert: "/home/jdoe/current/tls/certs/BrynMawr.crt";
        key: "/home/jdoe/current/tls/certs/BrynMawr.key";
    };
};
```

As this $gtmcrypt_config file contains both the database and tls groups, you need to set the gtm_passwd and gtmtls_passwd_<tlsid> environment variables using the maskpass utility. For more information on these environment variables, refer to "Environment Variables" (page 19). Note that in this $gtmcrypt_config file, verify-depth, CAfile, CApath, etc are OpenSSL options and PRODUCTION and DEVELOPMENT are 2 different tlsids which can be used for TLS replication and/or TLS sockets. By default the GT.M reference implementation plugin disables SSLv2, SSLv3, TLSv1, and TLSv1.1. You can selectively enable those protocols as needed (with **ssl-options: !SSL_OP_NO_SSLv2:!SSL_OP_NO_SSLv3:!SSL_OP_NO_TLSv1:!SSL_OP_NO_TLSv1_1**). Please note that OpenSSL, as compiled by the platform vendor, may also disable protocols.

OpenSSL Options

The gtmcrypt_config configuration file has two scopes for the following OpenSSL options. At the root of the TLS configuration, is a global configuration scope that applies to all tlsids listed in the configuration. Each tlsid can override the global configuration by redefining the same parameters. Any overridden option apply only to the specific tlsid configuration that defines it. The supported OpenSSL options are as follows:

CAfile When used in the tls level points to a file, in PEM format, describing the trusted CAs. The file can contain several CA certificates identified by sequences of:

```
-----BEGIN CERTIFICATE-----
... (CA certificate in base64 encoding) ...
-----END CERTIFICATE-----
```

Servers use this parameter, either specified for a tlsid in the configuration or with the WRITE / TLS command, verify certificates and to inform the client of acceptable certificate authorities. The determinant definition for the acceptable list of certificate authorities sent to the client comes in descending order of priority from the one specified by the WRITE /TLS("renegotiate",...) command, the one specified by the CAfile value in the tlsid section used to establish the TLS connection, and finally that specified at the tls level.

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 is an alternative to CAfile.

cipher-list   Specifies which cryptographic algorithms to use. The format of this option is described by the OpenSSL ciphers man page. An empty string uses a default value of "ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH" for replication connections and the OpenSSL default cipher list for socket connections.

crl   Points to a file containing list of revoked certificates. This file is created by the openssl utility.

format   Format of the certificate and private key pair. Currently, the GT.M TLS plug-in only supports PEM format.

cert   Path to the certificate.

key   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 gtmtls_passwd_<identifier>. Currently, the GT.M TLS plug-in only supports RSA private keys.

When placing the private key for a certificate at the beginning of the certificate file, you may omit the "key" item from the configuration file. The format of the combined file is:

```
-----BEGIN RSA PRIVATE KEY-----
[encoded key]
-----END RSA PRIVATE KEY-----
[empty line]
-----BEGIN CERTIFICATE-----
[encoded certificate]
-----END CERTIFICATE-----
 [empty line]
```

session-id-hex   Takes a string value which is used to set the SSL session_id context for server sockets, which may be specified in the tlsid section of a config file or on WRITE /TLS("RENEGOTIATE",...). See the OpenSSL man page for SSL_set_session_id_context for usage details. The value should consist of hexadecimal digits representing the desired value. Application code can call the %UTF2HEX utility routine to translate a character string to the corresponding string of hexadecimal digits. If neither the command or the associated tlsid section in the configuration file specify a session-id-hex option when creating the socket, GT.M uses the current tlsid, translated into hexadecimal digits.

session-timeout   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.

ssl-options   The ssl-options, documented in the man page for SSL_set_options, modify the default behavior of OpenSSL. When specifying multiple options, separate them with a colon (:) delimiter. The ssl-options specified in a labeled section add to, or override, those specified at the "tls" level. An exclamation mark ("!") preceding an option in a labeled section disables any default for that option specified at the tls: level; for example:

```
tls: {
 ssl-options: "SSL_OP_CIPHER_SERVER_PREFERENCE";
 mylabel: {
 ssl-options: "!SSL_OP_CIPHER_SERVER_PREFERENCE";
 };
 }
```

verify-depth	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.
	A verify-depth option specified in a labeled section applies to connections associated with that section.
verify-level	The verify-level option takes a string value to specify any additional certificate verification in addition to the basic OpenSSL verification. The only value currently accepted is "CHECK" which requests additional checks on the results of the basic OpenSSL certificate verification. A leading exclamation mark ("!") disables a verify-level option. The verify-level options specified at lower levels are merged with those options already specified at higher levels. CHECK is enabled by default for all TLS connections.
verify-mode	The verify-mode option specifies how OpenSSL verifies certificates. If no verify-mode is specified, it defaults to SSL_VERIFY_PEER. See the man page for SSL_set_verify for details. SSL_VERIFY_PEER has two additional flags which modify verification only for the server role; when adding them to the option string, use the colon (:) delimiter.

Generating demo TLS certificates

The following sections contains the steps for quick-start creation/signing of certificates for running the Database Encryption, TLS Replication, and/or TLS sockets examples. These steps are for demonstration/training purpose only and should never be used in production. The steps you might use in your environment and the certificate operations related to creation, signing, renewal, revocation, placement, protection, etc. should be a part of a comprehensive security plan that aligns with the security needs of your organization.

Creating a demo CA (Certification Authority)

Creating a demo root certificate authority involves three steps. In the real world, you would use certificates signed by competent Certificate Authorities (CAs). Certificate Authorities' certificates themselves are signed (and trusted) by other CAs eventually leading to a Root CA, which self-signs.

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.
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 and signing demo leaf-level certificates

The root certificate is used to sign regular, leaf-level certificates. Below are steps showing the creation of a certificate for authentication.

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:

```
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, Certificate Revocation Lists, and so on.

Example: Creating demo certificates

This section contains an example for quick-start creation/signing of demo certificates for use with running the TLS Replication, Database Encryption, and TLS sockets examples in the GT.M manuals. The OpenSSL configuration file pointed to by the

OPENSSL_CONF environment variable is example101.cnf. This configuration file is only suitable for the running examples and should not be used in production. Note that GT.M does not use openssl.cnf or the configuration file pointed to by the OPENSSL_CONF environment variable for any operation. The OpenSSL configuration settings that you might use in your environment and the certificate operations related to creation, signing, renewal, revocation, placement, protection, etc. should be a part of a comprehensive security plan that aligns with the security needs of your organization.

Create the following files:

cert_setup

This file sets up a dummy $PWD/certs directory for use in running TLS examples.

Here is the code:

```
echo "Creating cert directories ...in $PWD"
mkdir -p $PWD/certs/newcerts
touch $PWD/certs/index.txt
touch $PWD/certs/index.txt.attr
echo "01" > $PWD/certs/serial
echo "Generating root CA...."
./gen_ca
```

gen_ca

This file creates a demo root certification authority in the $PWD/certs directory for use with the TLS examples.

Here is the code:

```
#Generates root certification authority and sets it to expire in 365 days. Ensure that you have a properly
▶ configured /etc/ssl/openssl.cnf file.
mkdir -p $PWD/certs
openssl genrsa -des3 -out $PWD/certs/$1ca.key
openssl req -new -x509 -days 365 -key $PWD/certs/$1ca.key -out $PWD/certs/$1ca.crt
#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.
```

gen_leaf

This file creates the leaf-level certificate and gets it signed from the demo root certification authority.

Here is the code:

```
#Generates leaf-level certificates in $PWD/certs
openssl genrsa -des3 -out $PWD/certs/$1.key
openssl req -new -key $PWD/certs/$1.key -out $PWD/certs/$1.csr
openssl ca -config $PWD/example101.cnf -in $PWD/certs/$1.csr -out $PWD/certs/$1.crt
openssl x509 -in $PWD/certs/$1.crt -dates -issuer -subject -noout
```

example101.cnf

This file specifies the OpenSSL configuration file to use for running the example:

```
HOME = .
RANDFILE= $ENV::HOME/.rnd
[ ca ]
default_ca = CA_default

[ CA_default ]
dir  = ./certs
certs  = $dir/certs
crl_dir  = $dir/crl
database = $dir/index.txt
unique_subject = no
new_certs_dir = $dir/newcerts
certificate = $dir/ca.crt
serial  = $dir/serial
crlnumber = $dir/crlnumber
crl  = $dir/crl.pem
private_key = $dir/ca.key
RANDFILE = $dir/private/.rand
x509_extensions = usr_cert
name_opt  = ca_default
cert_opt  = ca_default
default_days = 365
default_crl_days= 30
default_md = default
preserve = no
policy  = policy_anything

[ policy_match ]
countryName  = match
stateOrProvinceName = match
organizationName = match
organizationalUnitName = optional
commonName  = supplied
emailAddress  = optional

[ policy_anything ]
countryName  = optional
stateOrProvinceName = optional
localityName  = optional
organizationName = optional
organizationalUnitName = optional
commonName  = optional
emailAddress  = optional

[ req ]
default_bits  = 1024
default_keyfile  = privkey.pem
distinguished_name = req_distinguished_name
attributes  = req_attributes
x509_extensions = v3_ca
string_mask = utf8only
```

```
[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = US
countryName_min = 2
countryName_max = 2
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Philadelphia
localityName = Collegeville
0.organizationName = Organization Name (eg, company)
0.organizationName_default = Example Pvt. Ltd.
organizationalUnitName  = Organizational Unit Name (eg, section)
organizationalUnitName_default = Example Unit
commonName = Common Name (e.g. server FQDN or YOUR name)
commonName_max = 64
emailAddress  = Email Address
emailAddress_max = 64

[ req_attributes ]
challengePassword  = A challenge password
challengePassword_min  = 4
challengePassword_max  = 20
unstructuredName  = An optional company name

[ usr_cert ]
basicConstraints=CA:FALSE
nsComment   = "OpenSSL Generated Certificate"
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer
basicConstraints = CA:true
```

Now make gen_leaf, gen_ca, and cert_setup executable and run the following commands:

```
export OPENSSL_CONF=$PWD/example101.cnf
./cert_setup
./gen_leaf demo
```

Enter blank values or example data as appropriate. You demo certificates are now ready for use in the documentation examples.