GT.M V5.5-000

Supplementary Instance Replication

Technical Bulletin

Contact Information

GT.M Group
Fidelity Information Services, Inc.
2 West Liberty Boulevard, Suite 300
Malvern, PA 19355
United States of America

GT.M Support for customers: +1 (610) 578-4226
gtmsupport@fisglobal.com
Switchboard: +1 (610) 296-8877
Website: http://fis-gtm.com

Legal Notice

Revision History		
Revision 1.1	20 March 2012	Improvements and corrections in "Upgrade Replication Instance File" (page 18) and "New Qualifiers" (page 21) sections.
Revision 1.0	February 22, 2012	First published version.

Table of Contents

Summary

Effective V5.5-000, FIS GT.M adds a new type of replication, called Supplementary Instance (SI) replication. The previous sole replication mechanism is now called Business Continuity (BC) replication in order to distinguish between them.

At the time of writing, V5.5-000 is the only GT.M release to support SI replication. Should later releases exist when you peruse this technical bulletin, please read "V5.5-000 or later, as clarified or qualified by the release notes for your release" for "V5.5-000" below.

In this document, unless more specifically designated, the term "transaction" refers to all TP transactions - updates within TSTART/TCOMMIT brackets (with nested transactions, the outermost one), and all so-called "mini transactions" - individual updates (Sets, Kills, and ZKills) not within TSTART/TCOMMIT brackets.

Motivation for Supplementary Instance Replication

GT.M replication was originally intended to provide business continuity for systems of record. For example with instances named **Ardmore** and *BrynMawr*, business logic processed on instance **Ardmore** can be streamed to instance *BrynMawr* so that should **Ardmore** ever go down, *BrynMawr* can immediately take over and provide continuity. In order to ensure that *BrynMawr* produces results consistent with **Ardmore**, *BrynMawr* can contain only material state information that is also in **Ardmore**.[1] To help ensure this consistency, GT.M BC replication prohibits locally generated database updates on a replicating secondary instance. Because they are conceptually part of atomic updates from the originating primary instance, GT.M triggers on replicating secondary instances are computed and committed locally.

Now consider an application such as running a report. This may require database updates including calculating and storing aggregates, statistics and interim results, or perhaps just to record the fact that the report was run. With traditional BC Replication, to offload such reporting from the originating primary instance is not a simple matter of running it on a replicating secondary instance, since updates are not permitted on replicating secondary instances.

SI replication allows replication from an instance (either **Ardmore** or *BrynMawr* in the example above) to another originating primary instance, Malvern. Malvern can execute its own business logic, computing and committing its own updates to its database, while receiving a replication stream. In turn, BC Replication from Malvern can provide it with its own replicating secondary instance, *Newtown*. In this example, only originating primary instances **Ardmore** and Malvern can execute business logic and compute database updates. Replicating secondary instances *BrynMawr* and *Newtown* are only permitted to receive and apply replication streams from their originating primary instances.

SI replication is intended to be a general purpose mechanism whose utility includes applications such as real-time decision support, warehousing, analytics and auditing.

[1] Non-material information in the database may be information that identifies the instance, local process ids, system load, etc.

Theory of Operation

This section presupposes an understanding of GT.M database replication and Logical Multi-Site (LMS) application configurations.

In order to create applications that remain available in the face of system and network failure, GT.M replication is asynchronous, which in turn means that the source and receiver ends of a replication connection are at an identical state only when there is no activity underway. To maintain consistency, and to restore it when restarting a replication connection, instances maintain a common, mutually coherent, instance-wide serial number called a journal sequence number. Since replication deals with an entire global variable name space, regardless of the mapping of global variables to database files, all updates participate in this numbering, even when modifying different database files.

When an originating primary instance and a replicating secondary instance connect, they establish the latest shared transaction (that is, the latest transaction they both have in common in their databases). If the instance configured as the secondary has any transactions in its database subsequent to that last transaction shared with the instance that is to be the primary, it rolls its database back to that last common transaction before beginning replication. The transactions that are rolled back go into an Unreplicated Transaction Log (previously known as a Lost Transaction File, a name we are retaining so as to not cause regressions). Identifying the common point for this rollback requires mutually coherent transaction ordering. Journal sequence numbers, while providing coherent transaction for BC replication do not suffice for SI replication.

To augment this capability for SI replication, journal sequence numbers are tagged with their origin- 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. Thus, of the journal sequence numbers in Malvern say 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 Malvern whereas stream # 1 indicates those transactions were generated in **Ardmore**. It is important to note that if, as part of restarting replication with **Ardmore**, Malvern needs to roll {101,1,18} off its database, database update serialization also requires it to roll {102,0,11} off as well, and both will appear in the Unreplicated Transactlon Log.

The same transaction that has a Malvern sequence number of 101 will have a different sequence number of 18 on **Ardmore** and *BrynMawr*. That is, the journal sequence number on **Ardmore** becomes the stream sequence number on Malvern. The replication instance file in Malvern contains information that allows GT.M to determine this mapping, so that when Malvern rolls {101,1,18} off its database, **Ardmore** knows that Malvern has rolled off **Ardmore**'s transaction 18.

If Malvern in turn implements BC replication to another instance *Newtown*, the tagging is propagated to *Newtown*, such that if **Ardmore** and Malvern both go down (e.g., if they are co-located in a data center that loses electricity), *BrynMawr* and *Newtown* can take over the functions of **Ardmore** and Malvern respectively, and *Newtown* can perform any synchronization needed in order to start accepting a replication stream from *BrynMawr*. as being a continuation of the updates generated by **Ardmore**, and *BrynMawr* in turn accepts *Newtown* as the successor to Malvern.

Types of Sequence Numbers

SI replication adds additional types of update related sequence numbers to those already in GT.M. This is an attempt to summarize in one place what they are and how they are related.

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. Thus, V5.5-000 introduces the concept of a stream sequence number.

On an originating primary instance that is not the recipient of an SI replication stream, the journal sequence number and the stream sequence number are the same.

On a primary instance that is the recipient of an SI replication stream, the journal sequence number continues to 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 (i.e., with values 1 through 15) and the journal sequence number for the transaction on the system from which it was replicated. These stream sequence numbers are propagated to downstream replicating secondary instances.

Stream sequence numbers are 64-bit unsigned integers.

Examples

To make the following scenarios easier to understand, each update is prefixed with the system where it was originally generated and the sequence number on that system and any BC replicating secondary instances.

Simple Example

The three systems initially operate in roles O (Originating primary instance), R (BC Replicating secondary instance) and S (recipient of an SI replication stream).

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

Ardmore	*BrynMawr*	Malvern	Comments
			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.
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

Ardmore	*BrynMawr*	Malvern	Comments
			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*, M39a, M40a, *B63*	Malvern operating as a supplementary instance to *BrynMawr* replicates transactions processed on *BrynMawr*, and also applies its own locally generated updates. Note that M39a & M40a 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.
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.

Ardmore	*BrynMawr*	Malvern	Comments
-	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

Ardmore	*BrynMawr*	Malvern	*Newtown*	Comments
				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*, *N74*, *B62*, *N75*, *B63*, *N76*, *B64*	While *BrynMawr* and *Newtown*, keep the enterprise in operation, the first data center is recovered. Since **Ardmore** has 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

GT.M V5.5-000 also introduces a mupip journal -rollback -backward -online feature that allows a database to be rolled back in state space *while an application is in operation*. This functionality is discussed in the release notes. Consider the following example where **Ardmore** rolls back its database.

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

Ardmore	***BrynMawr***	**Malvern**	**Comments**
			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**, M39, M40, **A97**a, M41, **A98**a, M42	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 both the original **A97** and **A98** as well as **A97**a and **A98**a. **A99** was never replicated to Malvern - **Ardmore** rolled back before it was replicated, and **A99**a has not yet made it to Malvern (it will soon, unless **Ardmore** rolls back again)

Limitations

SI replication is only supported on POSIX editions of GT.M - in other words, it is not supported by GT.M on OpenVMS. Furthermore, with this release, GT.M no longer supports replication between OpenVMS and POSIX platforms, or on POSIX platforms with GT.M releases prior to V5.1-000. To upgrade to GT.M V5.5-000, or to upgrade from GT.M on OpenVMS to GT.M V5.5-000 on a POSIX platform, first upgrade to GT.M V5.1-000 or later on a POSIX platform as an intermediate step.

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.

Procedures

Overview

Although V5.5-000 supports only one source stream for SI replication, the architecture allows for fifteen externally sourced streams (numbers 1 through 15), with stream 0 being locally generated updates. Under normal conditions, on a supplementary instance, one will see updates from streams 0 and 1. If the recipient of an SI stream has been moved from receiving replication from one source to another, you may see other stream numbers (corresponding to updates from other streams) as well.

When adding SI replication, the rules to remember are that (a) both source and receiver sides of SI replication must be V5.5-000, (b) upgrading an instance to V5.5-000 requires a new replication instance file because the replication instance file format for SI is not compatible with those of prior releases and (c) the -updateresync qualifier requires the name of a prior replication instance file when both source and receiver are V5.5-000.

Remember that except where an instance is an unreplicated sole instance, you should upgrade replicating secondary instances rather than originating primary instances. Starting with BC replication (e.g., **Ardmore** as originating primary and *BrynMawr* as replicating secondary), the simplest steps to start SI replication to Malvern are:

- Bring *BrynMawr* down and upgrade it to V5.5-000. *BrynMawr* requires a new replication instance file. Please refer to the relevant release notes for details of upgrading database files and global directories; unless otherwise instructed by FIS, always assume that object and journal files are specific to each GT.M release.

- Resume BC replication from **Ardmore** to *BrynMawr*. Since **Ardmore** is at an older GT.M release than V5.5-000, when starting the Receiver Server for the first time at *BrynMawr*, the -updateresync qualifier does not require the name of a prior replication instance file.

- Create supplementary instance Malvern from a backup of *BrynMawr* or **Ardmore**, if that is more convenient. Malvern will require a new replication instance file, created with the -supplementary qualifier.

- Start SI replication from *BrynMawr* to Malvern. Since Malvern and *BrynMawr* are both V5.5-000, the -updateresync qualifier used when the Malvern Receiver Server starts for the first time requires the old replication instance file copied, perhaps as part of a BACKUP, up from *BrynMawr* as its value.

At your convenience, once *BrynMawr* is upgraded you can:

- Switchover so that *BrynMawr* is the originating primary instance with BC replication to **Ardmore** and SI replication to Malvern. This is unchanged from current LMS procedures. SI replication from *BrynMawr* to Malvern can operate through the switchover.

- Bring **Ardmore** down and upgrade it to V5.5-000. It requires a new replication instance file.

- Start BC replication from *BrynMawr* to **Ardmore**. Since **Ardmore** and *BrynMawr* are both V5.5-000, the -updateresync qualifier for **Ardmore**'s first Receiver Server start requires the name of a prior replication instance file. As it cannot use **Ardmore**'s pre-V5.5-000 format replication instance file, in this special case, use a backup copy from *BrynMawr* as that prior file.

Upgrade Replication Instance File

To upgrade the replication instance file, perform the following steps:

- Shut down all mumps, MUPIP and DSE processes except Source and Receiver Server processes; then shut down the Receiver Server (and with it, the Update Process) and all Source Server processes. Use MUPIP RUNDOWN to confirm that all database files of the instance are closed and there are no processes accessing them.
- Rename the existing replication instance file after making a backup copy.
- Create a new replication instance file (you need to provide the instance name and instance file name, either with command line options or in environment variables, as documented in the Administration and Operations Guide):
  - If this is instance is to receive SI replication (Malvern in the examples above) or to receive BC replication from an instance that receives SI replication (*Newtown* in the examples above), use the command:

```
mupip replicate -instance_create -supplementary
```

  - Otherwise use the command:

```
mupip replicate -instance_create
```

- Prepare it to accept a replication stream:
  - Start a passive Source Server using the -updok flag.
  - Start the Receiver Server using the updateresync flag, e.g.: mupip replicate -receiver -start -updateresync=filename flag where filename is the prior replication file if the source is V5.5-000 and no filename if it is an older GT.M release (with other required command line flags, as documented in the Administration and Operations Guide).
- Start a Source Server on a root or propagating primary instance to replicate to this instance. Verify that updates on the source instance are successfully replicated to the receiver instance.

The -updateresync qualifier indicates that instead of negotiating a mutually agreed common starting point for synchronization the operator is guaranteeing the receiving instance has a valid state that matches the source instance currently or as some point in the past. Generally this means the receiving instance has just been updated with a backup copy from the source instance.

A GT.M V5.5-000 instance can source a BC replication stream to or receive a BC replication stream from older GT.M releases, subject to limitations as discussed in the Limitations section of this document. It is only for SI replication that both source and recipient must both be V5.5-000.

Creating a Supplementary Instance

A supplementary instance cannot be the first or sole instance that you upgrade to V5.5-000 - you must already have created an instance running V5.5-000 to provide a replication stream to the supplementary instance.

You can create a supplementary instance from (a) a backup copy of another instance, a supplementary instance, an originating primary or replicating secondary by giving it a new identity, or (b) a freshly created, new instance. An instance used to create a supplementary instance must already be upgraded to V5.5-000.

Starting with a backup of another instance, follow the procedures above under Upgrade Replication Instance File using the -supplementary flag to the mupip replicate -instance_create command.

Creating a supplementary instance from a backup of an existing instance creates an SI replication instance with all the database state in the existing instance and is perhaps the most common application need. But there may be situations when a supplementary instance only needs new data, for example if it is to provide a reporting service only for new customers, and historical data for old customers is just excess baggage. In this case, you can also create a supplementary instance by creating a new instance, pre-loading it with any data required, enabling replication for the database files (since an Update Process will only apply updates to a replicated - and journaled - database region), and following the procedures above under above under Upgrade Replication Instance File using the -supplementary=on flag to the mupip replicate -instance_create command. Ship a replication instance file from the source to provide the -updateresync=filename qualifier required when starting the Receiver Server for the first time.

Starting / Resuming Replication

For starting SI replication on an originating primary supplementary instance (Malvern in the above example), the procedure is similar to a non-supplementary instance except that one also needs to start a receiver server (after having started a source server to set up the journal pool) to receive updates from the corresponding non-supplementary instance (Ardmore or BrynMawr in this case). Similarly, as part of shutting down the originating primary supplementary instance, an added step is to shut down the Receiver Server (if it is up and running) before shutting down the Source Server.

Remember that for GT.M replication, the Receiver Server listens at a TCP port and the Source Server connects to it. If the Receiver Server is not ahead of the Source Server, replication simply starts and streams updates from the source to the receiver. When the Receiver Server is ahead of the Source Server, the cases are different for BC and SI replication.

For either BC or SI replication, if the Receiver Server is started with the new -autorollback qualifier, it performs an online rollback of the receiving instance, so that it is not ahead of the originating instance, creating an Unreplicated Transaction Log of any transactions that are rolled off the database. When started without the -autorollback qualifier, a Receiver Server notified by its Source Server of a rollback, logs the condition and exits so an operator can initiate appropriate steps.

For SI replication the new -noresync qualifier tells the Receiver Server not to rollback the database even if the receiver is ahead of the source. In this case, the Source Server starts replication from the last journal sequence number common to the two instances.

Changing the Replication Source

When changing the source of a supplementary replication stream to another in the same family of instances (for example in the example where **Ardmore** and Malvern crash and *Newtown* is to receive a replication stream from *BrynMawr*, start the Receiver Server normally (with either -autorollback or -noresync, as appropriate) and allow *BrynMawr* to connect to it. Instead of using -autorollback, you can also perform a mupip journal -rollback -backward -fetchresync before starting the Receiver Server.

To migrate a supplementary instance from receiving SI replication from one set of BC instances to a completely different set - for example, if Malvern is to switch from receiving SI replication from the set of {**Ardmore**, *BrynMawr*} to a completely unrelated set of instances {Pottstown, Sanatoga}

MUPIP Commands

SI replication adds and modifies qualifiers to the MUPIP REPLICATE and MUPIP JOURNAL commands.

Modified qualifiers

```
-updateresync=filename
```

Used when starting a Receiver Server on an instance where the replication instance file has no history records (usually because the instance file was newly created). The -updateresync qualifier now requires the file name of a replication instance file if the source of the replication stream is running GT.M V5.5-000 or higher, but not if the source is running a GT.M release prior to GT.M V5.5-000. This helps the Receiver Server by using history records from the input instance file (instead of the receiver instance file which has no history records) to exchange history information with the source and ensure both instances are in sync before safely starting replication. Previously there was no verification of history records in case of -updateresync usage and this opened up a safely hole in GT.M replication. For example, in the case where a receiver of an SI replication stream is started for the first time (the receiver and source have not communicated since the receiver replication instance file was created), this qualifier requires the name of a replication instance file (which is not the current replication instance file of the secondary instance) that is obtained from a backup of the source instance (taken at the same time the source databases were backed up). In this case, the Receiver Server startup should happen only after ensuring the backed up source database has been loaded onto the receiver instance. Once the receiver (started with -updateresync) and source connect, the Update Process logs a message containing the string "New History Content". The -updateresync should no longer be used for future Receiver Server startups. Note that the updateresync qualifier is not upward compatible from older releases of GT.M that never required a file name. Although FIS tries hard to keep GT.M upward compatible, our judgment is that this incompatibility represents the best option for adding the new SI replication functionality as well as enhancing the robustness of existing BC replication. Note that this qualifier was needed only in uncommon situations in GT.M V5.5-000 and prior releases. V5.5-000 make the need even more uncommon by eliminating it in the case where the instance has been recreated using a backup of another instance. This is because of a change to mupip replic -editinstance command which now supports a -name qualifier. For more details, refer to GT.M V5.5-000 Release notes.

New Qualifiers

```
-autorollback
```

Used when starting a Receiver Server. The command mupip replicate -receiver -start -autorollback allows the receiving instance of SI or BC replication to roll back as required when the source instance rolls back with a mupip journal -rollback -backward command. For more information on this flag, refer to GT.M V5.5-000 Release notes.

As autorollback uses mupip online rollback under the covers, it should be considered field test grade functionality as long as that function is considered field test grade functionality.

```
-noresync -
```

Used when starting a Receiver Server. The command mupip replicate -receiver -start -noresync instructs the Receiver Server to accept a SI replication stream even when the receiver is ahead of the source. In this case, the source and receiver servers exchange history records from the replication instance file to determine the common journal stream sequence number and replication resumes from that point onwards. Specifying -noresync on a BC replication stream is disallowed with a NORESYNCSUPPLONLY error. Use of -noresync on a SI replication stream receiver server where the receiving instance was started with -UPDNOTOK (updates are disabled) is disallowed with 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.

```
-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).

```
-rsync_strm=<strm_num>
```

Used when starting a rollback command with the -resync qualifier. The command mupip journal -rollback -resync= -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.

```
-supplementary
```

Used when creating a replication instance file. The command mupip replicate -instance_create -supplementary creates a replication instance file suitable for use in a supplementary instance. To create a replication instance file that is suitable for use in a non-supplementary instance, use the same command but without the -supplementary qualifier.

```
-updnotok
```

Used when starting a Source Server. The command mupip replicate -source -start -updnotok instructs the Source Server to not allow local updates on this instance. This is a synonym for the already existing -propagateprimary qualifier but is named so it better conveys its purpose.

```
-updok
```

Used when starting a Source Server. The command mupip replicate -source -start -updok instructs the Source Server to allow local updates on this instance. This is a synonym for the already existing -rootprimary qualifier but is named so it better conveys its purpose.

Error Messages and Recovery

See the Error Messages and Recovery Guide and V5.5-000 Release Notes.