Oracle® GoldenGate for Mainframe

Administration Guide

Version 10.0

October 2009

Administration Guide, version 10.0

Contents

PREFACE

About the GoldenGate Guides

The complete GoldenGate documentation set contains the following components:

Oracle® GoldenGate for Mainframe

- *Administration Guide:* Introduces GoldenGate components and explains how to plan for, configure, and implement GoldenGate on the NonStop platform.
- *Reference Guide:* Provides detailed information about GoldenGate parameters, commands, and functions for the NonStop platform.

Windows and UNIX platforms

- *Installation and Setup guides*: There is one such guide for each database that is supported by GoldenGate.
- *GoldenGate for Windows and UNIX Administrator Guide*: Introduces GoldenGate components and explains how to plan for, configure, and implement GoldenGate on the Windows and UNIX platforms.
- *Reference Guide*: Provides detailed information about GoldenGate parameters, commands, and functions for the Windows and UNIX platforms.
- *Troubleshooting and Tuning Guide:* Provides suggestions for improving the performance of GoldenGate in different situations, and provides solutions to common problems.

Typographic conventions used in this manual

This manual uses the following style conventions.

- Parameter and command arguments are shown in upper case, for example:
  ```
  CHECKPARAMS
  ```
- File names, table names, and other names are shown in lower case unless they are case-sensitive to the operating system or software application they are associated with, for example:
  ```
  account_tab
  GLOBALS
  ```
- Variables are shown within < > characters, for example:
  ```
  <group name>
  ```

- When one of multiple mutually-exclusive arguments must be selected, the selection is enclosed within braces and separated with pipe characters, for example:

  ```
  VIEW PARAMS {MGR | <group> | <file name>}
  ```

- Optional arguments are enclosed within brackets, for example:

  ```
  CLEANUP EXTRACT <group name> [, SAVE <count>]
  ```

- When there are numerous multiple optional arguments, a placeholder such as `[<option>]` may be used, and the options are listed and described separately, for example:

  ```
  TRANLOGOPTIONS [<option>]
  ```

- When an argument is accepted more than once, an ellipsis character (...) is used, for example:

  ```
  PARAMS ([<requirement rule>] <param spec> [, <param spec>] [, ...])
  ```

Conventions used in writing this manual

Conventions have been used in writing this manual in the following instances.

- References to NSK Enscribe **files** and SQL **tables.**

  The following terms have been used interchangeably in this manual and apply to both except where explictly noted.

  - File and table
  - Record and row
  - Field and column

- Ampersands in parameter file examples.

  Ampersands (&) have not been included in most parameter file examples. They are, however, required for all parameters that span multiple lines except those that must end with a semicolon.

New features in version 10

Parameters

- Prior to this version Audserv logged a message without abending when the security check omitted a file. The default behavior is now to log a message and abend (NOABENDONSECURITYCHECK). A new NOABENDONSECURITYCHECK option of AUDSERVPARAM allows this to be set back.

- The AUTOSTART parameter can now be used to trigger Manager to start Logger processes.

The the syntax has changed to:

```
AUTOSTART [<group type>] {<group name> | <process name>}
          [, ALLPROCESSES]
```

Where: <group type> can now include LOGGER:

<process name> is a required entry for the LOGGER <group type> used for the Logger process name in the format $xxnnn. Wildcards may be used for all or part of the name.

- AUTORESTART can now be used to trigger Manager to restart Extract and Replicat processes that have failed.

  The syntax is:

  ```
  AUTORESTART <process type> <group name>
  [, RETRIES <max retries>]
  [, WAITMINUTES <wait minutes>]
  [, RESETMINUTES <reset minutes>]
  ```

  Where: RETRIES sets the maximum number of times that Manager should try to restart a process before aborting retry efforts. The default number of tries is 2.

  WAITMINUTES is the amount of time to pause between discovering that a process has terminated abnormally and restarting the process. The default delay is 1 minute.

  RESETMINUTES is the window of time during which retries are counted. The default is 20 minutes. After the time expires, the number of retries reverts to zero.

- CHECKPOINTSECS parameter can now be used to control the frequency of routine checkpoints in Replicat as well as Extract.
- The Syncfile DUP parameter has new options:
  - INCLUDEFILECODE to specify file codes to be included in the DUP
  - EXCLUDEFILECODE to specify file codes to be excluded from the DUP
  - NAME to assign a logical name to the DUP
- The new FILEAGEDAYS parameter specifies the number of days a file can be inactive before Extract, Replicat, or Audserv ages it off its file list. Audserv inherits this parameter from the Extract.

  The syntax is

  ```
  FILEAGEDAYS <days>
  ```

  Only files added with a wildcard match are eligible to be removed. Files cannot be aged off if they are designated in FILE and MAP statements that have filters, SQLEXEC statements, or column mappings that use column functions.

- The new Manager parameter IPINTERFACE can be used to restrict the Manager process to the interface specified by an input IP address or DNS name.

  ```
  IPINTERFACE {<ip address> | <dsn name>}
  ```

- STATOPTIONS is a new parameter with the option NOZEROSUPPRESS to trigger reporting zero counts for updates, inserts and delete operation in Extract and Replicat. For Replicat it also allows PROGSTATS to report the time spent creating a file, purging a file, or ending a transaction.

  The syntax is:

  ```
  STATOPTIONS [ZEROSUPPRESS | NOZEROSUPPRESS]
            [, PROGSTATS] (Replicat only)
  ```

  The default is to suppress zero counts and to not report on the Replicat timings.

- The new TALUSEREXIT parameter can be used to call custom TAL routines at various points in Extract and Replicat processing.

- SUPPRESSMARKERMESSAGES stops messages being generated when markers are processed.

  For Extract and Replicat, the syntax is:

  ```
  SUPPRESSMARKERMESSAGES
  ```

  For GLOBALS, the syntax is:

  ```
  SUPPRESSMARKERMESSAGES {YES | NO}
  ```

- The EXITPARAM "exitparam string" option of FILE and MAP is no longer limited in size.

GGSCI commands

- Version 10 Logger processes include a new HOTSWAP option for SEND LOGGER. HOTSWAP instructs Logger to change to the specified object file.

  The syntax is:

  ```
  SEND LOGGER [<process name>]
     [HOTSWAP <object name>]
  ```

  If a Logger is running at the time of migration from a GoldenGate version 10 or higher to another version, the MIGRATE process will prompt to see if there should be a HOTSWAP of the Logger object.

  Note that using HOTSWAP to change the Logger program must be coordinated with the manual update of BASELIB.

Column conversion functions

- The GGFILEHEADER option has been added to @GETENV to return attributes of the trail file header record sent to NonStop from open systems. Attributes that can be returned include the trail version, file sequence number and size, information on the first and last records, and information on the local environment.

Functionality

- User exits can now be written in TAL as well as in C or COBOL. The TAL syntax for each of the callback functions has been added to the User Exits chapter of the Reference Guide.

- A new function is available to user exits. GET_USER_TOKEN_VALUE allows the value of a token to be retrieved.

- The function, GET_EXITPARAM_VALUE, is now available to user exits. It retrieves the value of an oversized string (over 256 bytes) entered with the EXITPARAM option of a FILE or MAP statement.

- Primary key updates coming from open system databases into a NonStop Enscribe or SQL/MP database are supported as of Version 10. These will be replicated by deleting the target record and inserting a new record with the new primary key.

  Primary key updates are still not supported for entry-sequenced and queue files.

  To continue to discard primary key updates instead of processing them, contact *GoldenGate Technical Support*.
- Version 10 includes support for HP Blades.

- The default location where GoldenGate libraries, such as BASELIB and GGSLIB, look for the AUDCFG segment file can now be assigned when running BUILDMAC or NLDLIB.

- A new library, PCREATE, intercepts a C runtime operating system function to allow the creation of high pin processes.

CHAPTER 1

Introducing GoldenGate for HP NonStop

GoldenGate for NonStop provides real-time transaction data management across the enterprise. To handle large transaction volumes, GoldenGate provides modular architecture and flexible processing to support heterogeneous applications and platforms. This chapter introduces GoldenGate architecture and features in the following topics:

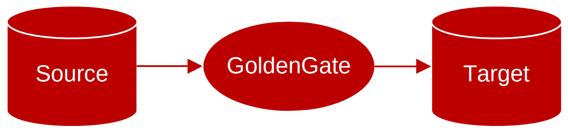
GoldenGate overview

GoldenGate can be configured to manage data from multiple, heterogeneous sources and targets. It also contains features that enable businesses to manage data at the transaction level across the enterprise. This section introduces both the configuration and the features of GoldenGate for NonStop.

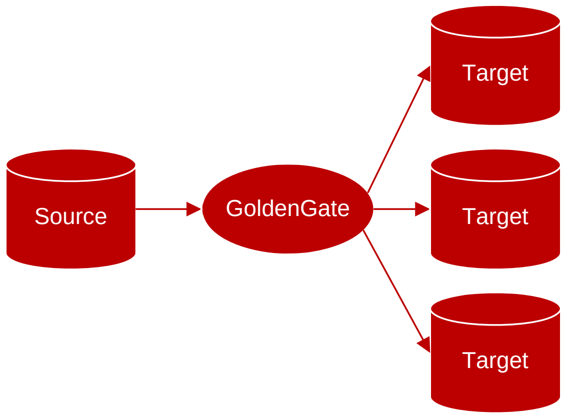
GoldenGate configuration

GoldenGate offers flexibility in how you configure and operate your transaction management system, supporting homogeneous and heterogeneous data replication. This allows you to configure GoldenGate to capture and deliver data as best suits your operating environment. Options include:

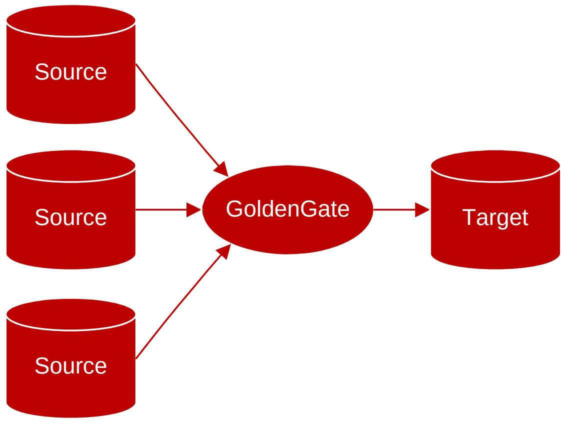
1. One-to-one, from a single source to a single target.



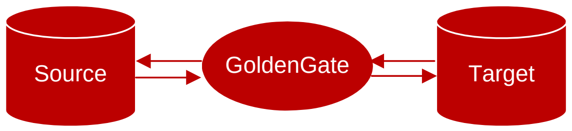
2. One-to-many, from a single source to multiple targets.



3. Many-to-one, from multiple sources to a single target.



4. Bi-directional, between a single source and a single target.



In doing so, the following business needs can be met:

- Change synchronization, supporting both online and batch change synchronization for Transaction Management Facility (TMF)-enabled and non-TMF-enabled applications.
    - Online change synchronization continuously processes incremental data changes.
    - Batch change synchronization processes change records that are generated during specific periods of time.

- Initial load, extracting complete records directly from a source file or table, then loading them into the target. You can use initial load to populate the target and to synchronize the source and target for change synchronization.
- Data distribution, sending extracted records to more than one target.

GoldenGate features

GoldenGate features allow you to select, map, and transform data so it is easily used for a variety of applications. You can configure GoldenGate to integrate and convert data by selecting data based on filtering criteria, as well as implement custom logic so GoldenGate works seamlessly with your own application portfolio.

For example, you can configure GoldenGate's activities by:

- Configuring data selection to deliver only required records, filter records to extract specific column data, and control which types of operations are extracted.
- Mapping named source files and tables to the target when the target has similar formats but different file or table names, split single rows into multiple rows, and combine rows from different tables to a single table.
- Implementing data transformations to convert dates from one format to another, perform arithmetic calculations, and transform DML operations, such as converting delete operations into insert operations on the target table for auditing purposes.

You can also configure GoldenGate to invoke your own custom programs and frequently-used routines by means of user exits, macros, and obey files. These features increase GoldenGate's flexibility by:

- Inserting user exits to call your applications and/or custom data management logic.
- Using macros to implement multiple uses of a statement, consolidate multiple commands, and invoke other macros.
- Automating frequently-used routines by using the OBEY command, which instructs GoldenGate to execute parameters specified in another parameter file.

Each of these features is enabled by GoldenGate's modular architecture that allows you to implement just the components you require. These components are introduced in the next section.

GoldenGate architecture

GoldenGate processes data by capturing records from a data source, housing it temporarily, then delivering it to a data target. Each of these steps is handled by a major component of GoldenGate's modular architecture.

GoldenGate components

GoldenGate for NonStop consists of the following components:

- Extract, for extracting and processing records from Transaction Monitoring Facility (TMF)-enabled applications.
- Logger, for extracting and processing records from non-TMF-enabled applications.
- Collector, for facilitating the transmittal of records between local and remote systems.

- GoldenGate trails, for transmitting change records from the source to the target.
- Replicat, for processing and replicating records to a target.
- GoldenGate Manager, for controlling, monitoring, and reporting on GoldenGate processing.
- Checkpoint groups, for helping maintain data integrity by tracking where, on the source, processing starts and stops.
- Parameters, that allow you compile instructions for Extract, Replicat, Manager, and utilities.
- Reader, for monitoring GoldenGate trails for distributed network transactions and communicating status information to the Coordinator.
- Coordinator, for tracking distributed network transactions to coordinate processing across multiple nodes.

Extract

Extract extracts source data from the TMF audit trail and writes it to one or more files, called GoldenGate trails. Multiple Extracts can operate on different sources at the same time. For example, one Extract could continuously extract data changes from a database and stream them to an up-to-date decision-support database, while another Extract performs batch extracts from other tables for periodic reporting. Or, two Extract processes can extract and transmit in parallel to two Replicat processes to minimize target latency when the databases are large.

Use Extract for:

- Initial load.
- Change synchronization for TMF-enabled applications.
- Transmitting change records between Logger (non-TMF) over TCP/IP to a remote target.
- Data distribution.

Logger

Logger performs data extracts when a NonStop source is non-TMF. Logger requires GGSLIB, an intercept library, that binds the GoldenGate application to the user's NonStop application. When an Enscribe operation (such as WRITE) executes, GGSLIB intercepts it and sends the record to Logger. Logger writes the records to a log trail which is read by Replicat.

Collector

When data is transmitted over a TCP/IP connection, the Collector resides on the target system and receives incoming records. Each Replicat group has a dedicated Collector process that terminates when the group's Extract process terminates.

Trails

Extract and Logger create trails to transmit data from the source to the target. A GoldenGate trail can contain a sequence of files or a single flat file. Generally, a GoldenGate trail is used during online change synchronization and a GoldenGate file is used for one-time tasks, such as initial data load or certain batch processes. There are two kinds of GoldenGate trails:

- Local trails. Local trails are transmitted to another NonStop system over Expand and read by Replicat. Local trails can also reside on the source and be used at a data source for Extract.
- Remote trails. Remote trails are transmitted to the target over TCP/IP and read by Replicat on the remote target.

Each record in a GoldenGate trail includes the data, a header with transaction information, an identifier of the record source, and other items. Trails are unstructured for best performance and are collected into transactions, which process in a continuous stream. Transaction identifiers indicate the first and last records in each transaction. Transactions are output in commit order, which guarantees that:

- Each record has been committed in the original database.
- All records in the original transaction are together in the output.
- Inserts, updates and deletes are presented, per record key, in the order in which they were applied.

To maximize throughput and minimize I/O load, extract data is written to, and read from, the trail in large blocks. By default, GoldenGate writes data to the trail in a proprietary format which allows data to be exchanged rapidly and accurately among heterogeneous databases. However, data can also be written in external ASCII, XML, or other formats compatible with different applications.

Replicat

Replicat reads data from GoldenGate trails that were created by Extract or Logger. You can run multiple instances of Replicat to read multiple GoldenGate trails. Replicat supports a high volume of replication activity on the target platform, transferring data in blocks rather than a single record at a time. SQL operations are compiled once and executed many times, and small transactions can be grouped into larger transactions to improve performance.

Manager

GoldenGate is managed by the Manager. Manager is responsible for starting and stopping Extract, Replicat, and their dependent subprocesses, such as Collector. Extract and Replicat checkpoints give Manager information about what resources are required at any given time. No other GoldenGate processing will run if Manager is stopped.

Syncfile

Syncfile allows you to schedule and manage file duplication when you wish to copy files in their entirety. This is a common requirement for maintaining a secondary GoldenGate instance that may see frequent database changes, but infrequent configuration file changes. However, Syncfile can copy any type of file, database or not, according to a schedule set by you. This makes it suitable for other off-hours, small scale copying tasks.

Processing Groups

To differentiate among multiple Extract or Replicat processes on a system, you define processing groups. A processing group consists of a process (either Extract, Replicat, or Syncfile), its parameter file, its checkpoint file (if applicable), and any other files associated with the process. For example, to replicate two sets of data in parallel, you would create two Replicat groups. You might name one group GGSDEL1 and the other GGSDEL2. Groups are defined by the ADD EXTRACT and ADD REPLICAT commands in GGSCI.

A group name can contain up to eight characters and is not case-sensitive. All files and checkpoints relating to a group share that name. Any time you issue a command to control or view processing, you supply a group name or multiple group names by means of a wildcard.

Checkpoints

Checkpoints are used to store the current read and write position of a process. They ensure that data changes marked for synchronization are extracted, and they prevent redundant extracts. Checkpoints provide fault tolerance by preventing the loss of data should the system, the network, or a GoldenGate process need to be restarted. For advanced synchronization configurations, checkpoints enable multiple Extract or Replicat processes to read from the same set of trails.

Checkpoints work with inter-process acknowledgments to prevent messages from being lost in the network. GoldenGate has a proprietary guaranteed-message delivery technology.

The Extract process checkpoints its position in the data source and in the trail. The Replicat process checkpoints its position in the trail. The checkpoint position is a combination of the sequence number of the trail file and the Relative Byte Address (RBA) of the trail.

The read checkpoint is always synchronized with the write checkpoint. Thus, if GoldenGate needs to re-read data that was already sent to the target system (for example, after a process failure), checkpoints enable accurate overwriting of the old data to the point where new transactions start and GoldenGate can resume processing.

Parameters

Parameters manage all GoldenGate components and utilities, allowing you to customize your data management environment to suit your needs. For example:

- The Manager parameter file contain instructions for controlling all other GoldenGate processes.
- The Logger parameter file contains instructions for capturing data from non-TMF applications.
- The Extract parameter file contains instructions for selecting, mapping, and transforming TMF data, and sending trails to Replicat.
- The Replicat parameter file contains instructions for selecting, mapping, and transforming data to the target.
- The Global parameter file contains instructions that can be applied globally to GoldenGate processing.

Reader

A Reader on each node scans the local GoldenGate trail for distributed transactions. When one is found, the Reader gathers local transaction information and sends it to the Coordinator process.

Coordinator

The Coordinator process receives information from each replicating node on the status of the distributed network transactions that are being processed. The transaction is not committed until the Coordinator has been notified that all of the updates have been received on their destination nodes. If any node has a failure, the changes are not applied.

GoldenGate processing

GoldenGate for NonStop processes data in a variety of ways, depending on your organization's needs and operating environment. This section introduces the primary ways GoldenGate captures and delivers data, including:

- Initial data synchronization
- Capturing data changes from TMF applications
- Using Extract for data distribution
- Batch processing
- Capturing directly from files
- Custom event processing

Initial data synchronization

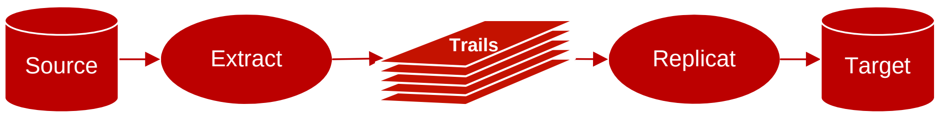
Run initial data synchronization to synchronize the source and target databases. This process can be run while your transaction system is operational because GoldenGate will not lock data when it captures and delivers records. With GoldenGate, your options for loading data include:

- Queuing data
- GoldenGate direct load.
- GoldenGate direct bulk load when the target is Oracle.

Queuing data

You can queue your data in one, or many, GoldenGate trails before loading your target for the first time. This lets you perform initial data synchronization while your transaction system remains online.

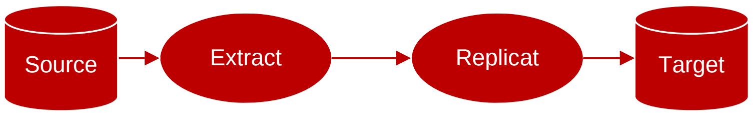
Figure 1 Queuing data



Direct load

Using GoldenGate direct load lets you extract data directly from source tables and send it in large blocks directly to Replicat, which writes data to its final target. This method is particularly effective for source data that does not need to be transformed (e.g. initial data loads).

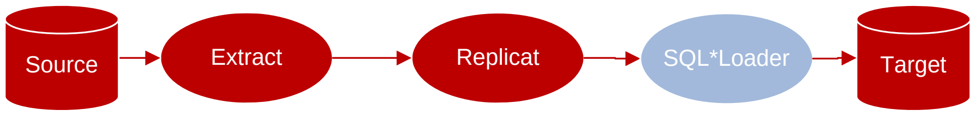
Figure 2 Direct load processing flow



Direct bulk load

If you are replicating from NonStop to Oracle, you can use GoldenGate direct bulk load. Using direct bulk load allows you to extract data directly from source tables and send it in a large block to the delivery process. Replicat then communicates directly to SQL*Loader. Using Replicat lets you perform additional data transformation prior to loading the data. The direct bulk load method is the fastest technique available when using GoldenGate.

Figure 3 Direct bulk load processing flow



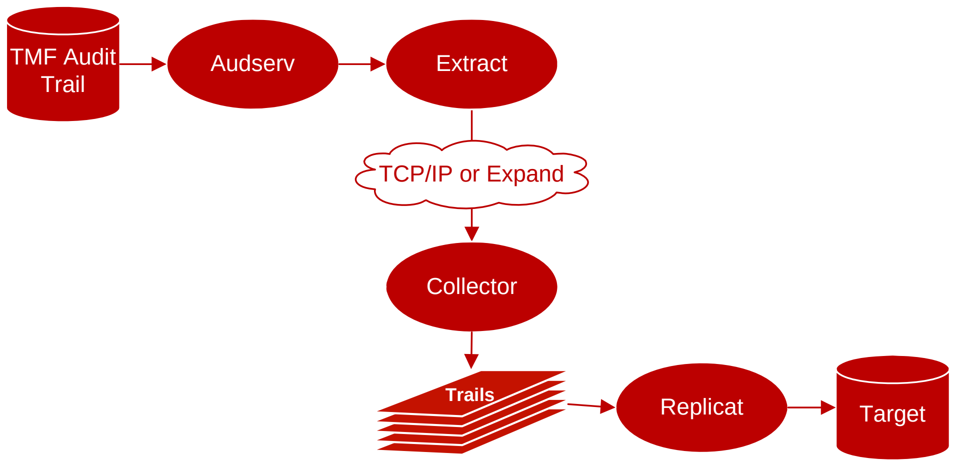
Capturing data changes from TMF applications

TMF audit trails provide the central resource for retrieving database changes in TMF-enabled applications. Changes to TMF-enabled Enscribe files and SQL tables are recorded in TMF audit trails for transaction integrity and recoverability. The following figure shows the processing flow for TMF-audited applications.

NOTE Because GoldenGate uses these audit trails for extract processing, plan and manage TMF-related activities carefully. GoldenGate's GGSCI and Manager tools provide optional audit management capabilities.

The Extract and Audserv work together to retrieve and process database changes for TMF applications. When started, Extract starts an Audserv process, which returns database changes from TMF audit trails. Audserv reads audit trails from their original location on disk, from a disk or tape dump, or from a user-specified alternate location. Audserv also determines the location of all required audit. Audserv can only return changes to tables or files if the user has read access.

Figure 4 TMF-audited process flow



Database changes include insert, update or delete operations, along with transaction information. Insert and update records are *after images*, or the format of the database record after the operation completes; delete records returned are *before images*. Before images for updates can also be returned.

Extract saves each image in memory until an associated transaction commit record is received. If the transaction aborts, the associated records are discarded. Committed records can be output to one or more user-designated extract files.

Audserv automatically excludes audit associated with SQL/MP and SQL/MX catalogs (file codes 564, 564, 565, 572, and 585).

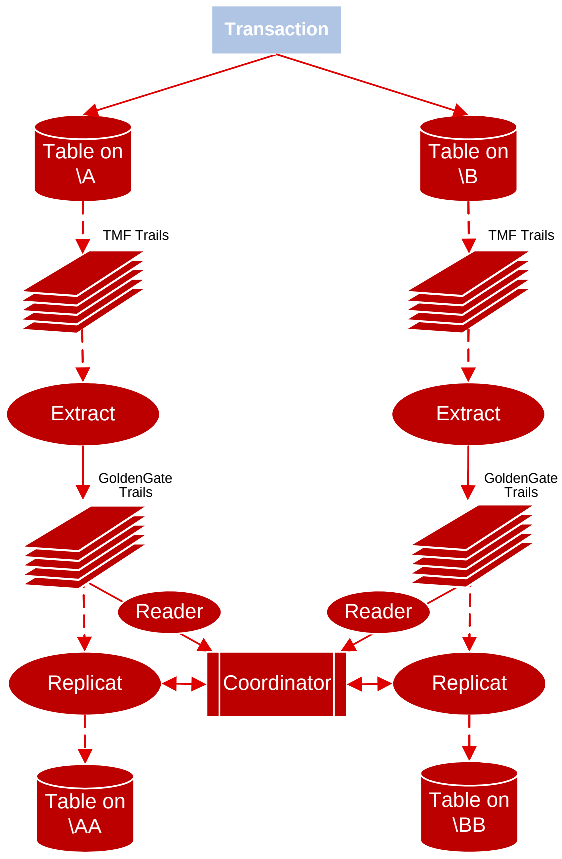
Capturing changes for distributed network transactions

In a multi-node environment a single transaction may update files across different nodes. GoldenGate includes processes to coordinate these network transactions so an outage on one of the nodes will not result in a partially updated transaction.

The central process is called the Coordinator. It receives information from each replicating node on the status of the changes being processed. The transaction is not committed until the Coordinator has been notified that all of the updates have been received on their destination nodes. If any node has a failure, the changes are not applied.

Figure 5 on page 23 illustrates the processing flow for a transaction across two nodes. An order from a customer, for example, may add information to the customer account file on the `\A` and the customer order file on `\B`, and these files may each be replicated to backup nodes.

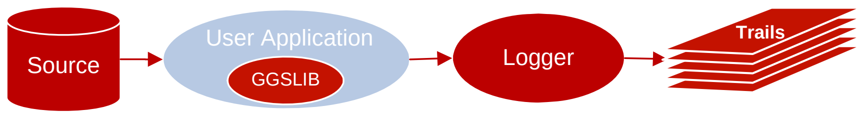
Figure 5 Processing flow for distributed network transactions



Capturing data changes from non-TMF applications

Many Enscribe applications do not use NonStop's TMF. GoldenGate provides an alternative method for capturing non-TMF-audited database changes. Figure 5 displays the processing flow.

Figure 6 GoldenGate processing flow—non-TMF applications



The GoldenGate Software intercept library (GGSLIB) is a group of functions with the same names as Guardian operating system procedures. GGSLIB is bound to Guardian, acting as an interface between application programs and NonStop.

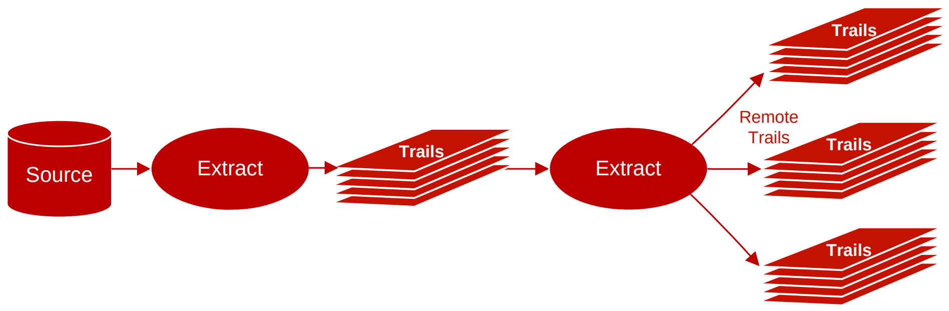
For example, when a Guardian function such as WRITE is called by an application, GGSLIB performs it instead of Guardian. The application is unaware of the substitution and executes, from an application programming standpoint, exactly as it did before.

If the function succeeds, it sends its data to Logger, which writes it to a GoldenGate log trail. Log trails are available for Extract and Replicat processing, which perform formatting, distribution and delivery steps.

Using Extract for data distribution

Extract can retrieve data from custom-created files or from trails created by Extract or Logger—in this sense, distributing data. Applications can take advantage of GoldenGate's data movement, formatting, conversion, and other features without reading the data directly from TMF audit trails or the database.

Figure 7 Extract as data distributor



Batch processing

When capturing incremental data changes in real-time is not appropriate, you can run batch processing. Batch runs process data generated during a specific time frame, defined by a begin and end time. Many GoldenGate operations, such as record selection, mapping, and field conversions can be performed during batch processing.

Capturing directly from files

In some situations, Extract can read directly from a file rather than from the log trail; however, the following conditions must be true:

- The file or sequence of files is *entry-sequenced*.
- Only *inserts* occur against the files—no updates or deletes.
- Records are inserted only at the *end of the file*.

Use this feature when:

- The method of logging is non-TMF-enabled.
- The files are BASE24 TLFX or PTLFX.
- The input files meet the conditions described above.
- You want to "trickle" the batch file contents throughout the day, rather than all at once at the end of the day.

Custom event processing

GoldenGate user exits make it possible to incorporate custom processing needs. A common application of user exits is database event triggering. For example, a user exit might contain code to page a supervisor when an account balance falls below a certain threshold. User exits reside outside the mainstream application—you can add, change, or remove them with almost no impact on the application.

GoldenGate commands

GGSCI is the command line interface that lets you interface with all GoldenGate components. Throughout this guide, you will see GGSCI commands described; they are your primary tools for configuring, operating, managing, and troubleshooting your data management environment.

To start GGSCI

Before you can start GGSCI, you must navigate to where GoldenGate is installed. Once you are in the correct subvolume, enter RUN GGSCI. Your prompt will change to a GGSCI prompt. For example:

```
TACL> VOLUME $DATA.GGS
TACL> RUN GGSCI
GGSCI>
```

Once you have reached the GGSCI command prompt, enter GGSCI commands on the command line as needed. With GGSCI commands, you can edit parameter files, add groups, view reports, and communicate with running processes. For more information, see the *Reference Guide*.

CHAPTER 2

Installing GoldenGate

Once you have planned how you want your data management environment to operate, you are ready to design its infrastructure and install GoldenGate. These procedures are covered in the following topics:

Installing GoldenGate

When you install GoldenGate for NonStop, you install GoldenGate components such as:

- Audserv
- GGSCI
- Manager
- Logger
- Extract
- Server Collector
- Replicat
- Coordinator
- Reader
- Syncfile
- DEFGEN
- DDLGEN
- MEAS utilities
- Supporting files, such as Help

Unlike earlier versions of GoldenGate, version 8.0 and higher auto-detects your Guardian operating system version and installs the native version of GoldenGate code where applicable. You may also tell GoldenGate to install TNS code during the installation routine. This procedure takes a short amount of time.

NOTE All HP NonStop Server installations for version 7 and above now use the UNPAK utility. Verify that you have a copy of this utility before proceeding.

To Install GoldenGate for NonStop:

1. Navigate to support.goldengate.com.
2. Select **Download**.
3. Select **HP NonStop Server**.

4. Select the version appropriate to your operating system.
5. Save the file to a directory on your workstation and unzip it to a temporary folder.
6. Upload the GGSUNPAK and GGS<OS version>PK files in BINARY mode to the volume and subvolume on your HP NonStop Server where you wish to install and run GoldenGate. We recommend that you preface the subvolume name with GGS and include the version of GoldenGate. For example, if you are installing GoldenGate version 10.0.0, then we recommend you install on $PROD.GGS1000.

   Note: If you are using GoldenGate against a TMF-enabled (or audited) database, then GoldenGate must be installed on a volume that is TMF-protected because the internal GoldenGate file must be audited.

7. Alter GGSUNPAK to be an Edit file.

```
TACL> FUP ALTER GGSUNPAK, CODE 101
```

8. Run GGSUNPAK and confirm the installation location. If the correct location is not displayed, you are prompted for it, as shown here.

```
TACL 2> run ggsunpack
Installing GoldenGate at $DATA4.CPSTEMP
Is this correct?(Y/N) N

Enter the volume and subvolume to install GoldenGate: $VOL.GGS1000
Installing GoldenGate at $VOL.GGS1000
Is this correct?(Y/N) Y
```

Text similar to the following will display:

```
UNPAK - File decompression program - T1255G06 - (2003-09-22)
Archive version: 1
File Mode RESTORE Program - T9074G07 (17NOV2005)
(C)2000 Compaq (C)2005 Hewlett Packard Development Company, L.P.
Drives: (\GGS2.$X6JZ)
System: \PROD Operating System: G06  Tape Version: 3
Backup options:  NO AUDITED, BLOCKSIZE 8, NO IGNORE, NO OPEN, PARTONLY
OFF, INDEXES IMPLICIT
Restore time: 20Apr2008 13:36  Backup time: 20Apr2008 13:33        Page:
1
Tape: 1     Code          EOF     Last modify   Owner RWEP   Type   Rec Bl

Summary Information
Disk related warnings = 2  Disk related errors = 0
Files restored = 91  Files not restored = 0

* Extractor-TDM and Replicator-TDM Installation *

*************************************************
*   The Installation routine has changed,       *
*   please pay attention to the prompting       *
*************************************************
```

9. (The prompts in this step appear only on a G06.16 operating system and are omitted on D30 or D40 systems.) For G06, you are asked if you need TNS versions of Extract and Replicat. GoldenGate recommends using the Native versions of these components, but if you have TNS mode user exits, enter Y at the prompts to install the TNS versions.

```
Since you are installing G06.16, we are installing Native mode objects.
Do you use TNS mode User Exits in Extract (Y/N)? N
Do you use TNS mode User Exits in Replicat (Y/N)? N
$DATA11.GGS1000.TEMP1 purged.
```

Note: For the purposes of this documentation, it is assumed that Native mode objects will be installed.

10. You are asked whether or not to build GoldenGate intercept libraries. GoldenGate recommends building a GGSLIB object, but it can be omitted if you do not need to capture non-audited Enscribe files or FUP-related activities. (You can build a GGSLIB object later.)

```
*************************************************
*   If you need the TNS mode GGS Intercept      *
*   Library GGSLIB then answer Yes,             *
*   Otherwise you may skip this step.           *
*************************************************
Build of new GGSLIB recommended. Build now (y/n)? Y
```

11. The BUILDMAC utility runs and searches for runtime libraries to bind with GGSLIB, and you may be prompted that one or more libraries is missing. Enter Y to continue.

```
*** Running BUILDMAC to build GGSLIB ***

*******************************************************
Looking for runtime libraries to BIND with GGSLIB
*******************************************************

Adding $SYSTEM.ZCOBOLRT.CLIBOBJ (COBOL74)
Adding $SYSTEM.ZCOB85RT.C8LIB (COBOL85)
$SYSTEM.ZCRERTL.CFELIB skipped (CRE)
Adding $SYSTEM.SYSTEM.CRELIB (CRE)
Adding $SYSTEM.SYSTEM.COBOLLIB (COBOL85)

One or more libraries was missing.  Continue anyway? (Y/N) Y
```

12. You are asked if you want to include a user library. Respond either **Y** or **N**.

```
Do you want to include your own User Library (y/n)? N
```

13. You are asked if you want to change the location of the AUDCFG.

```
Do you want to change the location for the AUDCFG segment (Y/N) : N
```

If you respond with yes, it prompts you for the new default location ($VOL.SUBVOL) of the AUDCFG segment.

Assuming you opted to build the intercept libraries, you will see the BIND steps being performed. You may see the following Binder messages that can be safely ignored. They are:

```
WARNING 60: Parameter mode mismatch on PXS_FILE_WRITE_parameter2.
Parameter mismatch between two external procedure declarations. The
parameters are declared as STRING and OTHER.
WARNING 61: Parameter type mismatch on PXFS_FILE_WRITE_parameter2.
Parameter mismatch between two external procedure declarations. The
parameters are declared as INT32 and OTHER.
WARNING 61 Parameter type mismatch on PXFS_FILE_WRITE_parameter3.
```

Other warnings may also be displayed and can be ignored.

14. You are asked if you want to build the native mode GoldenGate intercept libraries GGSLIBR and GGSSRL/GGSDLL. To install native code, enter **Y** and continue to the next step; otherwise, enter **N** to use TNS code and then skip to step 21.

```
*************************************************
*   If you need the Native mode GGS Intercept   *
*   Library GGSSRL/GGSLIBR then answer Yes,     *
*   Otherwise you may skip this step.           *
*************************************************

STOPPED: 2,33
CPU time: 0:00:00.945
1: Process terminated with warning diagnostics
STOPPED: 2,223
CPU time: 0:01:18.782
1: Process terminated with warning diagnostics
Build Native mode GGSLIBR & GGSSRL (Y/N?) Y
```

15. The NLDLIB macro runs, and you are asked if you want to include a user library. Reply either **Y** or **N**.

```
*** Running NLDLIB to build GGSLIBR & GGSSRL ***
----------------------------------------------------------------
NLDLIB builds the native relinkable libraries GGSLIBR and GGSSRL
       A Native mode GGSLIB needs the re-linkable versions of some
       Tandem SRLs. ZCREREL and ZCOBREL. If Native cobol support is not
       required then ZCOBREL may be omitted.
If Native Cobol support is required then you have to compile the
       Cobol program and search in GGSLIBR
A Native mode GGSSRL will be built for a User Library for Dynamic
linking

Enter X at any prompt to EXIT
----------------------------------------------------------------

Do you want to include your own User Library (Y/N) : N
```

16. You are asked if you want to change the location of the AUDCFG.

```
Do you want to change the location for the AUDCFG segment(Y/N) : N
```

If you respond with yes, it prompts you for the new default location ($VOL.SUBVOL) of the AUDCFG segment.

17. You are asked if you want to include ZCREREL. It is recommended to enter **Y** to include it unless you know your applications do not use these libraries.

```
Do you want to include ZCREREL (Y/N) : Y
```

NLDLIB searches for ZCREREL locations and displays them.

```
Looking for ZCREREL ..
$SYSTEM.SYS02.ZCREREL $SYSTEM.SYS03.ZCREREL $SYSTEM.SYS04.ZCREREL
$DATA09.R1269G09.ZCREREL $DATA09.R8431G09.ZCREREL
```

18. You are asked to enter the location of the ZCREREL to use.

```
Enter location of ZCREREL : $SYSTEM.SYS03.ZCREREL
```

19. You are asked if you want to include ZCOBREL. It is recommended to enter **Y** to include it unless you know your applications do not use these libraries.

```
Do you want to include ZCOBREL (Y/N) : Y
```

NLDLIB searches for ZCOBREL locations and displays them.

```
Looking for ZCOBREL ..
$SYSTEM.SYS02.ZCOBREL $SYSTEM.SYS03.ZCOBREL $SYSTEM.SYS04.ZCOBREL
$DATA09.R8107G09.ZCOBREL $DATA09.R8108D46.ZCOBREL
```

20. You are asked to enter the location of the ZCOBREL to use. If you plan to use COBOL, enter a location; otherwise, you can press **Enter** to bypass this prompt.

```
Enter location of ZCOBREL : $SYSTEM.SYS03.ZCOBREL
```

Note: If you omit a ZCOBREL location, you are prompted to confirm your choice:

```
Omit ZCOBREL (y/n):Y
```

NLDLIB builds the GGSLIBR and GGSSRL libraries, displaying a series of informational messages and the names of the files that were created.

Note: On systems running Guardian G06.23 or earlier, the GGSUNPAK utility installs and accelerates Audserv and Logdump. If you are running G06.24 or later, you will not see any acceleration messages.

21. You are asked if GoldenGate should use an existing SQL catalog or create a new catalog. Either enter **X** to specify no catalog or enter the name of a catalog to use. If you enter a catalog name and it does not exist, you are asked whether to create it.

```
SQL Catalog for Compilation (X for no catalog)? GGSCAT
Catalog $TRNG04.CPSCAT does not exist. Create it (Y/N)? Y
```

The SQL compilation completes, and then libraries and programs are linked. This happens automatically and does not require action from you.

22. You are asked if you want to migrate information from another GoldenGate environment. If you are installing GoldenGate for the first time, enter N. If you are upgrading from a previous version, see "Upgrading GoldenGate" on page 32. A migration requires shutting down existing GoldenGate processes and, if Logger is being used, the user application. (The following assumes a new installation.)

```
Would you like to Migrate information
from another GGS Environment (Y/N)? N
```

23. You are advised to license your programs as shown in the following prompt, and then the installation utility finishes.

```
Make sure to secure files appropriately. If EXTRACT is to be executed
outside of the SUPER group, you must PROGID and LICENSE the AUDSERV
program for SUPER.SUPER authority. PRIVLIB must also be licensed
if executing Replicat outside SUPER group.

Audserv/LOGDUMP licensing only applies to TMF installations.
TMFARUL2 must be licensed
The Native Audserv must be licensed (A TMFARUL2 restriction)
The Native Logdump must be licensed if you are going to use it
to read TMF trails (A TMFARUL2 restriction)

TACL> LOGON SUPER.SUPER
TACL> FUP
-GIVE Audserv, SUPER.SUPER
-SECURE Audserv, "NUNU", PROGID
-LICENSE Audserv
-LICENSE PRIV*
-LICENSE LOGDUMP
-LICENSE TMFARUL2

Installation Complete.
```

Licensing must be executed by SUPER.SUPER because the following privileged access is required:

- Audserv and TMFARUL2 access the TMF trails through audit read routines.
- PRIVLIB contains privileged Replicat code that invokes certain privileged FUP DUP or FUP LOAD operations.
- Logdump provides a version that can read the audit read routines. If you do not want Logdump to be able to read the audit trails, do not license it.

Note: If the version of your SQL catalog does not allow FUP to alter the program, you can PROGID and license the programs by using the following commands to license the components shown.

```
TACL> LOGON SUPER.SUPER
TACL> SQLCI
 >> ALTER PROGRAM AUDSERV OWNER 255,255;
 >> ALTER PROGRAM AUDSERV SECURE "NUNU" PROGID;
 >> EXIT
TACL> FUP
 - LICENSE (AUDSERV, PRIV*, LOGDUMP, TMFARUL2)
 - EXIT
```

24. Once your product is installed, log on as the owner of your GoldenGate environment.
25. (Optional) You may purge the `*PK` files, if desired.

Upgrading GoldenGate

Upgrading GoldenGate requires three phases: backup, installation, and migration.

- **Backup:** Perform a full backup of your GoldenGate environment before installing and migrating to the latest GoldenGate version.
- **Installation:** Install GoldenGate following the steps discussed in "Installing GoldenGate" on page 26**.** Keep in mind that you must specify a new subvolume for your upgraded GoldenGate instance.
- **Migration:** Once you have installed GoldenGate, you must migrate to the latest instance. Perform the following steps to complete the migration.

Migrating from a previous version of GoldenGate:

1. Bring down your current GoldenGate instance:

```
TACL> RUN GGSCI
GGSCI> STOP SYNCFILE *
GGSCI> STOP REPLICAT *
GGSCI> STOP EXTRACT *
GGSCI> STOP LOGGER
GGSCI> STOP MANAGER
GGSCI> EXIT
```

2. After you install GoldenGate, `GGSUNPAK` prompts you:

```
Would you like to Migrate information from another GGS environment (Y/N):
Y
```

3. Enter your previous GoldenGate volume and subvolume:

```
Enter old GGS vol.subvol: $PROD.GGS9506

Copying GGS data from $PROD.GGS9506...

File Utility Program - T6553G07 - (19DEC2005)    System  \GGS2
(C)1981 Tandem (C)2005 Hewlett Packard Development Company, L.P.
ALLOW ABENDS OFF
ALTER CONTEXT, NO AUDIT
LOAD $DATA11.GGS9506.CONTEXT, CONTEXT, SORTED
RECORDS LOADED: 8
ALTER CONTEXT, AUDIT
ALTER GROUP, NO AUDIT
COPY $DATA11.GGS9506.GROUP, GROUP
3 RECORDS TRANSFERRED
ALTER GROUP, AUDIT
ALTER EXTCTXT, NO AUDIT
COPY $DATA11.GGS9506.EXTCTXT, EXTCTXT
1 RECORDS TRANSFERRED
ALTER EXTCTXT, AUDIT
ALTER REPGRP, NO AUDIT
COPY $DATA11.GGS9506.REPGRP, REPGRP
1 RECORDS TRANSFERRED
ALTER REPGRP, NO AUDIT
COPY $DATA11.GGS9506.REPGRP, REPGRP
1 RECORDS TRANSFERRED
ALTER REPGRP, AUDIT
ALTER REPCTXT, NO AUDIT
COPY $DATA11.GGS9506.REPCTXT, REPCTXT
2 RECORDS TRANSFERRED
ALTER REPCTXT, AUDIT
ALTER REMCTXT, NO AUDIT
ALTER REMCTXT, AUDIT
ALTER RMTCTXT, NO AUDIT
COPY $DATA11.GGS9506.RMTCTXT, RMTCTXT
1 RECORDS TRANSFERRED
ALTER RMTCTXT, AUDIT
LOAD $DATA11.GGS9506.LOGGGS, LOGGGS, SORTED
RECORDS LOADED: 188
ALTER LOGCONF, NO AUDIT
ALTER LOGCONF, AUDIT
ALTER MRKRGGS, NO AUDIT
ALTER MRKRGGS, AUDIT
```

4. If you have a running Logger process, you are asked if you want to HOTSWAP the Logger object.

```
Logger was running during migrate
Do you want to HOWSWAP logger to the new ENV  (Y?N)
```

   If you answer yes, the program will change to the Logger object running in the migration process installation location.

   NOTE You can HOTSWAP only when migrating from a version 10 or higher environment. **Caution:** You must coordinate HOTSWAP with any manual steps to upgrade BASELIB as part of the upgrade process.

5. DUP any Obey files you use in your current GoldenGate environment. This step is particularly critical if your Obey files contain table define names, or startup/shutdown commands.
6. Recompile your user exits, then rebind them into GoldenGate. You must perform this step to ensure your user exits work with your newly-installed version of GoldenGate.

   NOTE When you install the new version of GoldenGate, all program locations are updated to the new default. If Extract and/or Replicat are expected to run from a specific place, you must manually point to the correct pointer using the ALTER command.

```
TACL> ALTER EXTRACT <group name> PROGRAM <program location>
```

7. Point your applications to the new version of GGSLIB.

```
TACL> BIND PROGRAMS, CHANGELIB
```

   Enter the list of applications you wish to bind.

Enter your new GGSLIB when prompted.

Enter GO to bind your new library to your applications.

8. Restart your GoldenGate components in your new environment.

Preparing your production environment

This section outlines considerations that impact the overall architecture of your particular GoldenGate environment. These considerations include:

- Sizing
- CPU requirements
- Disk space requirements
- Data communications requirements

Sizing

Before you begin running GoldenGate in your production environment, you should consider several factors regarding the size of your replication environment, including the following statistics.

- **Counts of inserts, updates and deletes generated on a per-file basis.** A common reference point is the application's transaction rate, from which these numbers can often be estimated.
- **The byte generation rate caused by inserts, updates and deletes.** Use this statistic to estimate disk space requirements and communications bandwidth. Byte generation totals during peak periods and over an extended time frame are also important.
- **For update-intensive applications, the number of bytes in a record that change during an update.** This is probably the most difficult statistic to extract (especially for Enscribe files) but can have a dramatic impact on bandwidth requirements. The number of bytes that change in a record during an update determines the compression rate for the record.
- **The cache hit rate and file busy percentages of the most heavily updated files.** This has a direct bearing on the throughput possible by individual Replicat processes on the target system.
- **Which programs are causing the heaviest amount of update activity.** It is often possible to avoid replicating certain types of activity (such as FUP LOAD) by excluding those programs from replication. Measuring individual programs is a concern primarily for non-TMF data processing.

GoldenGate Software provides two programs, MEASFLS and MEASRPT, to assist in determining sizing and configuration parameters for TMF and non-TMF based processing. MEASFLS gathers statistics using MEASURE. MEASRPT interprets MEASURE statistics and produces a report that assists in the sizing process.The MEASFLS and MEASRPT programs collect these statistics easily without requiring knowledge of the Measure utility.

Reducing overall requirements

In some cases, due to bandwidth or application constraints, it can be desirable to eliminate certain files, programs or processes from GoldenGate.

To get a view of overall replication activity excluding certain files or programs, use the MEASRPT EXCLUDEFILE and EXCLUDEPROGRAM parameters.

CPU requirements

CPU requirements for GoldenGate components vary primarily according to transactions rates, record sizes, and the I/O mix (numbers of updates versus inserts and deletes).

NOTE The following section details system performance statistics in a variety of situations. However, your operating environment, including operating system versions, hardware, and network topology will affect the performance you experience. The following figures should be taken only as guidelines, useful for reference when you are planning your GoldenGate implementation.

As a percentage of overall application processing, GoldenGate as a general rule requires more CPU resources when:

- I/O is a larger percentage of overall application processing.
- Files are buffered.
- TMF auditing is off.
- Cache hits are high.
- When TMF is on, audit compression is used.
- All files are replicated.

More power is required due to the fact that the application can produce relatively more I/O under these conditions.

TMF data extraction and transfer

GoldenGate extracts data in TMF applications by reading TMF audit trails (using the Audserv program). Audserv passes the data in large blocks to Extract, which formats and moves the data to the target system.

GoldenGate Software has observed TMF extraction takes between 1% and 10% of overall system processing. Typically this figure is 2-3%, but can skew higher when:

- the database is SQL.
- as a percentage, there are a high number of updates in the application.
- the SQL update statements set a large number of columns.
- AUDITCOMPRESS is on (turning AUDITCOMPRESS off is not recommended since this may have an adverse effect on performance).

Non-TMF data extraction and transfer

Non-TMF data extraction takes more processing power than TMF because blocks of data are output to log trails on disk, then transferred to the target system by Extract in a separate step.

It has been observed that non-TMF extraction activities take between 2% and 14% of overall application processing. This means that, if the application is running with an average of 50% busy for the CPUs, after replication is implemented the CPUs will be between 51% and 57% busy. This represents a significant variance and can be traced to the environmental factors listed above.

Intermediate GoldenGate trails

You can improve performance by writing data to intermediate trails. An intermediate trail is a trail written by Extract, extracted again, and transferred to Replicat. I/O to intermediate GoldenGate trails is serial, buffered and blocked at 28K bytes when the system is busy. This means that I/O rates upwards of three gigabytes per hour are possible to trails on the target (higher rates are possible when you are running multiple instances of Extract, IP channels and controllers). The impact of writing one gigabyte of data per hour using Extract has been measured to consume 5.5% of a single CPU on an S7000 system. (See the Note at the beginning of this chapter regarding system variance and its impact on performance).

Any impact resulting from intermediate trails is absorbed by the target system. The volume of data written to the target is likely to be far less than the corresponding audit generated on the source system for TMF applications.

To minimize contention with other disk activity, isolate trails on separate disk drives when possible.

Replicat performance

Replicat performance on the target system is related closely to the I/O performance on the source system, as I/O is essentially the same on both. This means you can estimate the impact of I/O on the target system by determining the File Busy percentage of application files and tables on the source system.

When the target database is TMF-protected, the Replicat program can improve performance by 2-3 times by using the GROUPTRANSOPS parameter. This parameter groups several source transactions into a single transaction on the target, improving the performance of TMF significantly.

Disk space requirements

Disk space requirements can vary depending on whether you are processing data from TMF-enabled applications or from non-TMF-enabled applications.

TMF data requirements

TMF replication requires temporary storage for data read from audit trails. In most scenarios, extracted data is output directly to the target system. TMF audit trails generally act as safety storage if the target system or link is unavailable or performing slowly.

In most cases, you must reserve enough space for TMF audit trails to tolerate an outage of one or more days. To determine your space requirements, check the rate at which audit is generated for a week or month and average the amount of space filled per day. GoldenGate can process tape dumps, so you can have a backlog as old as the oldest tape dump. However, restoring a tape dump can be cumbersome and usually requires manual intervention, so disk is preferred whenever possible.

Another way to reduce source system disk requirements is to store data in a local trail, then move it across the network through a separate Extract process. In many cases, the amount of data written to a trail by Extract is far less than the corresponding amount of data in the audit trails, especially when a subset of data is being replicated. Filtering unnecessary data in this manner reduces the total disk requirement.

Non-TMF requirements

Non-TMF database changes are stored in a log trail, which must be a local trail. Logged data is subsequently moved to the target system and replicated.

During normal activity, records in the log trail are replicated almost as soon as they become available, making the amount of disk required quite low. However, you must consider more extreme conditions when sizing disk space. For example, replication activity can fall behind due to:

- Target system maintenance (which might require suspension of Replicat processing).
- Slow data replication on the target due to other activity on the system.
- Slow or downed communications links between the source and target.

When any of these conditions arise, you need to store backlog locally. Many organizations use the general rule of accommodating one full day's worth of database updates. Allocate the number of bytes generated during this period for log trails on the source system.

In addition, you can move data to trails on the target system to minimize data loss caused by potential delays in Replicat processing. Size these trails for potential delays in replication.

When sizing for longer outages (for example, more than one day), you can use tape backups to reduce the amount of disk required.

Data communications requirements

Each record GoldenGate sends to the target system includes a header portion and a data portion. To compute total byte generation and transfer requirements in a given period, sum the byte counts of the headers and data for each record inserted, updated or deleted during the period.

The overall amount of activity can be calculated using the following statement. (Headers vary in size, however, the average is approximately 50 bytes. For this reason, the following example uses an 80-byte header.)

```
(number of inserts * (size of average insert + 80)) +
(number of deletes * (size of average record + 80)) +
(number of updates * (size of averaged compressed update + 80))
```

For Enscribe files, the size of insert and delete records can vary and may not be the same as the record size for the file (which represents the maximum size of each record). For inserts and deletes, all bytes in the record are transferred, along with a header.

For SQL tables, all bytes are transferred for insert and delete records as well.

With GoldenGate, the following data is not transferred to the target system:

- images of updates before the update occurs (only after images are sent for hot site backup)
- alternate key data
- SQL indexes
- data generated by FUP RELOAD
- assorted other TMF records

When sizing data communications bandwidth, peak byte generation is important. Bandwidth that can accommodate peak volumes helps guarantee low latency between the source and target databases at all times.

When moving data, Extract accumulates and sends records in 28K blocks across the network whenever database activity is high. As a result, the amount of data moved through Extract over a communication link can be approximately 75-80% of link capacity. For example, on a 10-megabit Ethernet network with approximately 4-megabits per second of true throughput, Extract can move approximately 3.5M bits per second (430KB/sec).

NOTE All network performance statistics are suggestions. Your own network configuration, bandwidth demand, and operating environment will determine the kind of performance you can expect. You can boost actual throughput considerably by using router-based compression between the source and target systems (a 4:1 ratio is fairly typical). This is in addition to update compression used by GoldenGate modules. For more information, contact your network vendor or NonStop representative.

When sizing communications requirements, note that GoldenGate can send data over multiple TCP/IP channels simultaneously. You can achieve total bandwidth requirements over multiple lines or networks as an alternative to a single path.

Preparing the source and target

Once you have addressed GoldenGate planning considerations, you are ready to meet its prerequisites. These include:

- Downloading source data definitions
- Synchronizing source and target

Downloading source data definitions

You can optionally download source data definitions from the source database into a definitions file on the target system. In most cases, you will want to download source data definitions to the target.

When the target resides on another NonStop system, run the DEFGEN utility to download data definitions. When the target resides on Windows/UNIX systems, run the DDLGEN utility to download data definitions.

Synchronizing source and target

Before running online synchronization for the first time, run an initial load to synchronize the source and target. This applies a copy of the source to the target. If Extract or Replicat stops running for any reason, you may need to re-synchronize the source and target and restart the Extract and Replicat groups to reset the checkpoints. For further assistance with your initial load, see “Configuring Initial Data Synchronization” on page 57.

CHAPTER 3
Configuring GoldenGate

Once you have installed GoldenGate, you must configure it to suit your own business needs. This chapter outlines techniques and procedures for the following configuration tasks.

Overview

Before running GoldenGate, you must make some decisions regarding your GoldenGate installation. This includes determining your required resources and their configuration. Other planning considerations include:

- Are you capturing change data from TMF-enabled or non-TMF-enabled applications?
- Are you transmitting data to targets over TCP/IP?
- Which topology configurations are you using?
- How much data communications capacity is required between the source and target systems?
- How much additional disk space is required to accommodate replication?
- Can GoldenGate accommodate present and future transaction volumes?
- How much overhead will GoldenGate add to the source and target systems?
- How can you scale GoldenGate to meet high volume requirements?

Configuring TMF-enabled processing

Extract and its servant program Audserv read TMF data in large blocks (upwards of 28K at a time) before transferring the data to a GoldenGate trail. This requires a small percentage of I/O messages because Audserv retrieves blocks of records from audit cache rather than from disk.

You must consider several factors when planning for Audserv to read TMF data. These include:

- Adding columns to a source table.
- Ensuring all audit is processed.
- Keeping necessary audit available for Extract.
- Minimizing vulnerability to outages.
- Configuring FUP RELOAD activity.
- Re-configuring TMF.

Adding columns to a source table

Occasionally columns are added to source database tables. This means when a layout of your file or table changes, you must stop Extract, make the changes, update any source definitions files provided by DEFGEN, then restart Extract so it retrieves the new definition from the dictionary or SQL catalog.

Ensuring all audit is processed

Various system events can require that you ensure all audit records are processed before the event occurs. Examples include system maintenance, (such as an operating system upgrade), TMF shutdown, and other events. Failing to do so can result in missed data.

There are several methods for verifying that Extract is current with TMF activity.

- Use the GGSCI SEND EXTRACT AUDITEND command to determine Extract's position in the audit trail. If the response indicates that all audit is processed, Extract has no more work to do assuming that TMF-related applications are down or idle.
- Use the GGSCI ADD MARKER command to insert a marker record into the audit trails after some significant event (such as taking the application down). Once Extract and Replicat have processed the marker, you can assume that all records prior to that point has been processed.
- Issue the INFO EXTRACT command from GGSCI, which returns the Extract lag (approximate number of bytes and time behind the audit trails). If the status is RUNNING and the number of bytes behind is less than 5000, it is likely that all audit has been processed.
- Issue a LAG EXTRACT command from GGSCI which reports the current lag times

Keeping necessary audit available for Extract

TMF purges audit trails it no longer requires, because it has no knowledge of outside processes that depend on it, such as Extract. This means you must plan how to keep audit trails available. This section discusses several options:

- Make sure a certain number of audit files are always available, either in production or backed up to an alternate subvolume.
- Copy the audit trails to an alternate subvolume (away from production TMF) and let Extract read them from the alternate location.
- Configure the audit trails to make disk dumps, and let Extract read them.
- Configure the audit trails to make tape dumps, and let Extract restore the audit.
- Include the DISKTHRESHOLD parameter in the Manager parameter file, so Manager warns you when audit trails are in danger of being purged.

Ensuring TMF cannot purge audit

Keep a certain number of audit files in production or as backup copies. One method for backing up files is using the GGSCI ADD ATCONFIG command with the DUPFILES option. Should you choose this option, limiting the number of duplicate files ensures that the backup disk does not fill up.

NOTE Using the DUPFILES option greatly increases the resources required to run GoldenGate. This is because duplicate audit requires exponentially more disk space.

Copying the audit to an alternate location

You can instruct Manager to copy audit trails to an alternate volume, then point Extract to read the alternate trails first. This keeps Extract activity from affecting production. To duplicate audit automatically, use the GGSCI ADD ATCONFIG command with the ALTLOC and DUPFILES or DUP options.

Using tape dumps as an alternate location

If you specify a tape as the alternate location, Extract displays a message asking the operator to restore the tape. The Extract program restores tape dumps to one of three locations before processing the audit. In order of preference, the locations are:

- The subvolume indicated by the ALTLOC option of the ADD ATCONFIG command.
- The first restore volume configured for the audit trail with TMFCOM.
- The original location of the file.

To preserve disk space the restored file is purged as soon as it is processed, unless the restore was performed before runtime. To prevent redundant restores, Extract determines if the restore occurs before runtime. If yes, Extract assumes other Extract groups may need the file and does not purge it. Manager purges them at the appropriate time if the ADD ATCONFIG PURGE option is set.

Restoring tape dumps before runtime can be convenient. To determine which tapes need to be restored for a specific Extract group, use the GGSCI STATUS EXTRACT command. The command lists the names of required audit files and whether they exist on disk or tape. All files on tape need to be restored. The GGSCI STATUS AUDITTRAIL command lists the names of all audit trails required across all Extract groups.

Minimizing vulnerability to outages

Extended network or target system outages can have an adverse impact on Extract processing. When the intended target system is unavailable, Extract cannot process the audit trail. If the target system remains down, critical audit will eventually be deleted from the system before it can be processed.

To prevent this problem, extract the data to a local trail for Replicat to access over Expand. This solution only applies when both the source and target are NonStop systems.

An alternative is to extract the data from the audit trails to an intermediate GoldenGate trail on the source, then configure a second Extract to move data to the target system. This ensures that data can always be extracted.

Outages also pose problems for transactions that are distributed across nodes. See “Configuring for distributed network transactions” on page 43 for information on ensuring transaction integrity for distributed transactions.

Configuring FUP RELOAD activity

FUP RELOAD commands are used to optimize database storage and access. They also generate a large amount of audit compared with normal activity. This can cause Extract to fall significantly behind the current location in the audit trails, sometimes requiring audit tape dumps to be restored. This process requires operator intervention.

You can often avoid tape restores by scheduling FUP RELOADs more effectively. Schedule reloads less frequently, or over several periods rather than all at once (for instance, reload 20% of the database each night for five nights, instead of reloading 100% of the database in a single evening).

Data compression

You can optionally configure GoldenGate to compress data before sending it over TCP/IP. The Collector automatically decompresses it on the target system. To compress records over TCP/IP, include the COMPRESS and COMPRESSTHRESHOLD options in the RMTHOST parameter statement.

- COMPRESS specifies that outgoing block of extracted changes are compressed, resulting in a typically 4:1 ratio or better.
- COMPRESSTHRESHOLD sets the minimum byte size for which compression will occur. The default is 1000 bytes.

For TMF-audited Enscribe files, set the NonStop AUDITCOMPRESS file attribute when creating the file. For non-TMF files, specify the COMPRESSUPDATES argument in the Logger configuration.

Compressed Enscribe records

Whether TMF or non-TMF, Enscribe compression transfers the following data (rather than sending all field values).

- Each fragment of the record that changed
- The key fragment of the record
- Four additional bytes per fragment indicating fragment position and length

The format of a compressed Enscribe record is as follows:

```
<field offset><field length><field value>...
```

Argument	Description
`<field offset>`	The offset within the original record of the changed value (2 bytes)
`<field length>`	The length of <field value> (2 bytes)
`<field value>`	The data, including null or varchar length indicators

The first field in a compressed Enscribe record is the primary or system key.

Compressed SQL records

By default, SQL updates are compressed in the audit trails. This means each SQL update record includes the following data.

- Each column that was SET in the SQL UPDATE statement
- Each key column in each row updated
- Four additional bytes per column indicating column number and length

Unlike Enscribe compression, you can estimate SQL update size directly using the MEASFLS and MEASRPT utilities and do not need other methods of estimation.

The format of a compressed SQL record is as follows:

```
<field index><field length><field value>...
```

Argument	Description
`<field index>`	The ordinal index of the SQL column within the source tables (2 bytes)
`<field length>`	The length of <field value> (2 bytes)
`<field value>`	The data, including null or varchar length indicators

DCOMPRESS file attribute not supported

Turn off the NonStop DCOMPRESS file attribute for both SQL tables and Enscribe files extracted using TMF audit trails. When DCOMPRESS is on, compression occurs within each data block, which prevents the resolution of entire record values. Extract is permitted, but unpredictable results can occur.

AUDITCOMPRESS file attribute considerations

When update operations occur on a file or table with audit compression on, only changed columns or fields, and those that are part of the primary key, are recorded. This means the full update images are not immediately available to Extract. Instead, a compressed image is retrieved and output.

For exact replication, this is acceptable because only changes are required. However, problems can occur in the following circumstances:

- A selection clause includes columns that are not part of the source file's primary key.
- Columns are mapped, and the primary key of the target is different than that of the source.
- User exits or custom applications do not anticipate compressed records, which are more complex to process.

Extract provides an option to retrieve full record images from the original database. However, retrieving each update can slow processing considerably. The options you invoke, and whether or not you use audit compression, is based on your application's requirements.

The NonStop AUDITCOMPRESS attribute is controlled at the file and table level using FUP and SQLCI.

Configuring for distributed network transactions

In a multi-node environment a single transaction may include changes to files on more than one node. For example, a customer's order may require updates to the customer file on \A, the customer account file on \B, and the order file on \C. Updates like these, as well as updates to tables that are partitioned across nodes, are referred to as *distributed network transactions.*

To help ensure the completeness of the transaction when one node experiences an outage, you should configure components that coordinate the updates for distributed network transactions. This avoids part of a transaction being committed while the changes going to a disabled node are lost.

To following processes play a part in this coordination. The required configuration set up is explained for each component.

- Manager

  When using a Coordinator, PURGEOLDEXTRACTS should be defined for the Manager rather than Replicat or Extract. This allows consideration of Coordinator checkpoints to ensure trail files are not purged before Coordinator has completed processing them. See "Recommendations for managing trail purges" on page 127.

  Also the Manager on the node where the Coordinator resides may optionally be configured to AUTOSTART the Coordinator process.

- Extract

  There are no configuration changes needed for Extract, but if it has the PURGEOLDEXTRACTS parameter, this should be moved to the Manager.

- Replicat

  The COORDINATOR parameter is added to the Replicat parameter file to define the name of the process that is coordinating its distributed transactions. When the Replicat encounters a distributed transaction, it communicates with this Coordinator to determine when it can process that transaction.

  If the Replicat has the PURGEOLDEXTRACTS parameter, it should be moved to the Manager to allow consideration of the Coordinator's checkpoints.

- Reader

  READER parameters are included in the COORDINATOR parameter file. These are used to configure Reader processes when the Coordinator is started.

  The Reader scans the local GoldenGate trail for distributed transactions. When one is found, the Reader gathers local transaction information and sends it to the Coordinator process.

  NOTE If Readers will not be configured because distributed network transactions do not need to be replicated, the Extract parameter EXCLUDEGGSTRANSRECS can be used. This will suppress the creation of trail records that track distributed network transactions.

- Coordinator

  A Coordinator process must be added on one of the nodes in the system. This is added using the GGSCI ADD COORDINATOR command. The parameter file for it includes READER parameters to establish the Reader process for each node and GoldenGate trail.

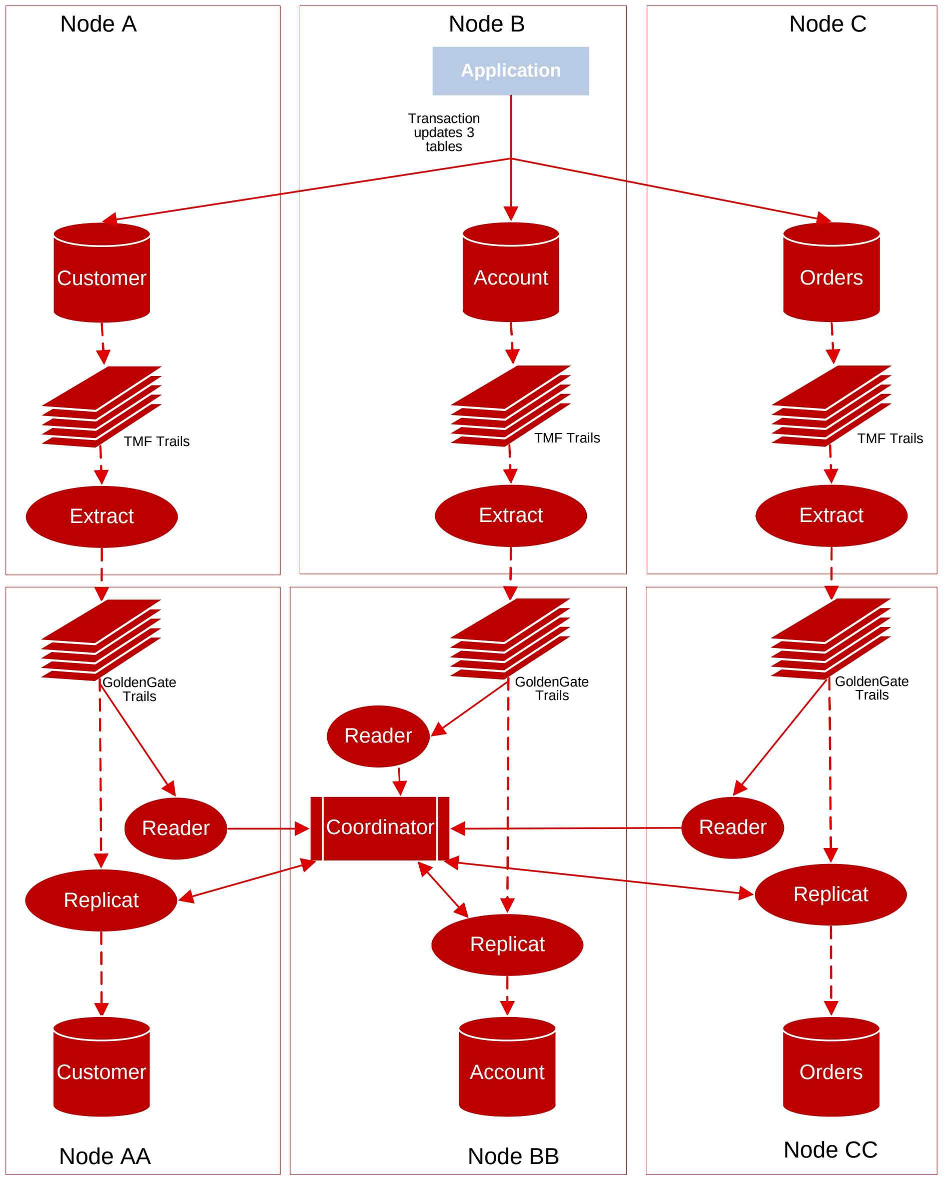
Figure 8 Sample Coordinator parameter file

```
COORDINATOR COORD1
FASTREADS
READER EXTTRAIL \NY.$DATA5.GGSDAT.AA, PROCESS $GGRD1, CPU 1, PRI 180
READER EXTTRAIL \LA.$DATA01.GGSDAT.BB, PROCESS $GGRD2
READER EXTTRAIL \FL.$DATA2.GGSDAT.CC, CPU 1, PRI 170
```

The Coordinator process receives information from the Readers, tallies the number of changes that have been received, and stores checkpoints. It uses this information to respond to queries from the Replicats on each of the nodes asking if the transaction is complete. When all of the operations for the transaction have verified their arrival, the Coordinator releases the transaction to the Replicats for processing.

The following diagram shows an example of coordination processes for a distributed network transaction that spans three nodes, with each node replicated to a backup node.

Figure 9 Process flow for distributed network transaction support



Re-configuring TMF

When facts about the audit trails change, the checkpoints recorded by Extract can be invalidated, and TMF must be re-configured.

Before re-configuring TMF:

1. Use the GGSCI INFO EXTRACT * command to ensure that all Extract groups have processed through the end of the last audit file.
2. Use the GGSCI DELETE ATCONFIG * command to delete the current audit management parameters.
3. Delete all Extract groups.

After TMF is reconfigured:

1. Manually re-add all of the Extract groups.
2. Purge audit files that were restored or copied to an alternate location.

Using TMFCOM, dynamically add and delete the volumes on which audit files are located. Deleting an ACTIVE or a RESTORE volume can have adverse effects. Before deleting a volume, make sure all groups have processed outstanding audit on that volume, or copy all files on that volume to the alternate location. After a volume is deleted, the Extract process and Manager will not be able to find the associated audit. You can add an ACTIVE or RESTORE volume with no impact on Extract operations.

Configuring non-TMF-enabled processing

To capture data from non-TMF applications, you must bind GGSLIB to the user application. GGSLIB will intercept certain NonStop commands in the application's place, while Logger will write data to a log trail. This causes the following planning issues:

- Maintaining data integrity
- Supported file types and operations
- System utilities that update databases
- Private memory and stack space
- Impact on existing application performance

Maintaining data integrity

The following issues can cause GGSLIB and Logger to miss records and/or compromise data integrity:

- Log processes are stopped by an operator while the application is updating a database. Several safeguards are built in to deal with this potential problem.
- If a log process is stopped from TACL by process number, which can happen accidentally, the backup process takes over with no loss of data.
- If a log process is stopped from TACL by name, this is assumed to be a mistake (since the proper method is the GGSCI STOP LOGGER command). Manager immediately restarts log processes stopped this way, although records can be lost if this occurs while there is activity in the system.

- Double CPU failure occurs, taking down both the primary and backup log process CPUs. When this happens, other data integrity issues will surface on NonStop as a whole, such as loss of file buffers.
- Application programs are not bound with GGSLIB. This can happen when a program is omitted from the initial bind list. This can also happen when migrating new object code into production, then forgetting to perform the GGSLIB bind. To avoid this problem, include GGSLIB binding into version control procedures and check any programs that generate warnings as explained in “Authentication for bound programs” on page 48.
- An application process is killed from TACL. This can mean that reads from or writes to the database could be lost in transit to the log process, depending on the timing of the STOP command. This is not a problem when issuing FREEZE and STOP commands to Pathway servers.
- Extract or Replicat processes fall far behind Logger. Eventually, log trails are recycled by Manager, regardless of whether or not they are required by Extract or Replicat. EMS warnings can be generated to alert operators to this condition. This most likely happens when a network or target system is down for an extended period of time.

Supported file types and operations

GGSLIB and Logger behave according to the following rules regarding file operations.

- The following file types are supported: Key-sequenced, entry-sequenced, queue-files, syskey-files, relative and unstructured file operations. However, updates to edit files and the spooler cannot be extracted. Unstructured files must be extracted explicitly (using the GETUNSTRUCTURED parameter in the Logger parameter file).
- Bulk I/O operations, i.e. operations that invoke SETMODE, are supported. The current list of SETMODEs includes:
    - 1 - Set file security
    - 2 - Set file owner
    - 3 - Set write verification
    - 57 - Set serial writes
    - 90 - Set buffered
    - 92 - Set maxextents
    - 93 - Set unstructured buffer length
    - 94 - Set auditcompress
    - 97 - Set licensed
    - 123 - Set generic lock key length
    - 138 - Set/Reset corrupt
    - 153 - Set variable length audit compression

  FUP DUP, FUP LOAD and SELECT n AREA in COBOL programs are also included.
- To extract bulk I/O operations, specify the GETBULKIO option in the Logger parameter file. FUP COPY is supported by default. Use GETFILEOPS in Extract and Replicat to propagate these operations to the target database.
- FILE ALTER, CREATE, DUP, PURGE, PURGEDATA, and RENAME operations (to disk files) are supported.

- The following CONTROL operations are supported:
    - 2 - set end-of-line
    - 20 - PURGEDATA
    - 21 - Allocate/Deallocate extents
- Use GETFILEOPS in Extract and Replicat to propagate the operations listed above to the target database.
- Undocumented, privileged function calls used by FUP DUP and FUP LOAD to change file labels are supported (this requires PRIVLIB to be licensed and included as Replicat's user library). These functions are required to fully implement FUP DUP and FUP LOAD of key-sequenced files.

Authentication for bound programs

An exit can be activated within NonStop's Safeguard to access the GoldenGate module SFGEXIT. This program runs as an independent process to monitor non-audited file opens for update access. (Opens for audited files or SQL tables and read-only opens are ignored.) When such an open is found, SFGEXIT determines if the opening program has the GoldenGate intercept library bound to it. If it does not, the following warning is issued to EMS to alert the user that updates may occur without replication.

```
GoldenGate Library is not bound to $<vol>.<subvol>.<program name> and it
may update $<vol>.<subvol>.<application file name>
```

System utilities that update databases

Standard NonStop utilities, notably FUP and TACL, perform file operations such as CREATE, COPY, PURGE, PURGEDATA, DUP, LOAD, and RENAME. You can monitor these activities by binding GGSLIB to these utilities just as you would to an application program.

Private memory and stack space

GGSLIB routines minimize stack space requirements. By doing so, programs are ensured that there will be enough stack room for normal activities.

For its own working space, GGSLIB allocates a small private memory segment to handle in-transit I/O buffers and keep its own state variables.

Impact on existing application performance

GGSLIB and Logger add a small amount of overhead to existing application activities. Messages to log processes are sent asynchronously (NOWAIT) to avoid making the application wait for logging to occur. In addition, log processes write all data sequentially into buffered files for the best possible performance.

Configuring GoldenGate global operations

GLOBALS parameter file

GoldenGate provides the GLOBALS parameter file to standardize GoldenGate configuration. Normally, you set global parameters when you install GoldenGate. Once set, you rarely need to change them. Some of the operations you can standardize are:

- The initial allocation for wildcard entries.
- The timeout value when GGSCI communicates with GoldenGate components.
- NonStop nodes in the network.
- The refresh interval.
- TACL defines for GGS_AUDCFG and GGS_PREFIX when not using the default.

To support versatility, some of the parameters set in GLOBALS can be temporarily overridden by other GoldenGate programs.

See the *Reference Guide* for more information about global parameters.

Changing the default location of AUDCFG

Run BUILDMAC or NLDLIB to change the default location where an instance of BASELIB, GGSLIB, GGSSRL, or GGSDLL will look for the AUDCFG segment. As it builds the new library, the macro prompts to ask if you want to change the AUDCFG location. If the answer is yes, you will be prompted for the new default $VOL.SUBVOL location.

If you want multiple GoldenGate environments to each have a different location for the AUDCFG segment, each environment will need a unique copy of GGSLIB or BASELIB linked with the location specific to that environment.

If the library specifies a different location for the AUDCFG than the DEFINES included in the GLOBALS parameters, the GLOBALS DEFINES will override the library.

Configuring replication

Replicat provides a high degree of flexibility when processing data between files; however, there can be logical restrictions involved for which you need to plan. This section details different scenarios that require additional planning, including:

- Replicating SQL tables with system keys
- Non-key entry-sequenced relative files and tables
- Load balancing and performance issues
- Potential problems with audit compressed files
- Conflicts with updating the target
- Many-to-one replication
- Bi-directional replication
- Replicating data to non-TMF enabled databases
- Replicating new SQL columns

Replicating SQL tables with system keys

Entry-sequenced SQL tables with non-unique keys are sometimes difficult to replicate accurately. This is because their keys are a SYSKEY value generated by the system. Replicat has no control over the SYSKEY value when replicating an insert operation into the target table; therefore, subsequent update and delete records cannot be replicated exactly. Even though the SYSKEY value of the original record is known, the replicated record has a different

SYSKEY value, requiring you to create a workaround so your keys resolve properly.

There are two methods for working with this issue. You can specify a view that contains all columns from the base table excluding the SYSKEY. Use the view as the target in the replication MAP, along with a KEYCOLS specification to define a different method for accessing the table for delete and update operations. This requires each target row to have some type of unique identifier, such as a unique index.

Another method is to add a column called GGS_SYSKEY to your target table, then map the source SYSKEY value to the GGS_SYSKEY column. Specify GGS_SYSKEY in the KEYCOL option of the map argument and use the FORCEUSESYSKEY parameter.

Replicating primary key updates

Although Nonstop Enscribe and SQL/MP do not allow changes to primary keys, operations for primary key updates may be received from GoldenGate systems running for other databases. To maintain compatibility, GoldenGate for NonStop processes these primary key update operations by deleting the record and then inserting it with the same data but a new primary key.

Primary key updates for Enscribe entry-sequenced and queue files are not supported and will generate an error.

The default is to process primary key updates, but a parameter is available to turn this off and discard the record. Contact *GoldenGate Technical Support* to use this parameter.

Missing row errors

Because values are needed for the columns that were not changed, an error will occur if the record cannot be fetched from the target database.

If HANDLECOLLISIONS is turned on and the fetch fails, there will be an attempt to insert the missing record. Otherwise if REPERROR responses have been defined for a missing row, the rules specfied by the REPERROR will be applied.

Non-audited target

An error message is returned if an unaudited Enscribe record is deleted and then the insert of the new primary key record fails. Since it is not possible to backout the records processed since the last checkpoint, the system will advance the checkpoint to the record that is in error. User intervention will be required to correct the target record and restart the Replicat.

- For a file system error, correct the cause of the problem and insert the record from the discard file. Then skip over the primary key update record by advancng the checkpoint RBA to the next record.
- If the insert generates a duplicate error, try to determine if the discarded record is more correct than the target record. If it is, delete the record in the file and replace it with the discarded record. Then skip over the primary key update record by advancng the checkpoint RBA to the next record.

Files and tables other than key-sequenced

You can replicate files and tables that are not key-sequenced, but there will be conditions that apply.

For relative files, GoldenGate forces the relative key of the target file to be the same as the source, so target records can be found for updates and deletes. The condition is that you can only replicate from a single source to a single target.

You have more flexibility if the relative file or table has a unique index. Then the columns in that index can be specified with KEYCOLS to identify a path for update and delete statements. However, any application that stores system keys as foreign keys in other tables will have unreliable results.

For entry-sequenced files or tables, selective replication (that is, where selection criteria are applied) is only feasible for inserts. This is due to the difficulty identifying the correct target record for updates. Selective replication from one source to one target is feasible for relative files and tables.

Entry-sequenced files can be replicated in the same order when the source database is TMF audited because the TMF data is in the correct order. If the source database is non-TMF, and GGSLIB is used to extract the data, records may be output in the target file in a different order than they appear in the source. This has a corresponding effect when updates to entry-sequenced records are processed: the record address of the source may be different from that in the target, resulting in a missing or incorrect update.

To get around this, when replicating a non-TMF entry-sequenced file from one source to one target, you can use the parameter and option ENTRYSEQUPDATES EXACTKEY. This requires the target file to be opened with PROTECTED or EXCLUSIVE access so other processes (including other Replicats) can not update the file. See the *Reference Guide* for more information on how to use this parameter.

See “Bi-directional replication” on page 53 for information on an environment not limited to single source updating a single target.

Load balancing and performance issues

Replicat often proves to be a bottleneck when initially configured, especially for hot site applications that replicate the entire database. This bottleneck is due to the fact that Replicat often mimics the original application’s processing. In general, this may mean many more random, unbuffered I/Os. In contrast, Extract and Logger perform serial, buffered I/Os, usually in large blocks.

To solve this problem, configure multiple Replicat processes, each of which replicates a portion of the overall data.

One way to do this is assign different files or tables to different Replicat processes. This is conceptually simple. For example, if an application consists of data in four tables, TAB1, TAB2, TAB3, and TAB4, let Replicat process #1 replicate TAB1 and TAB2, while Replicat process #2 replicates TAB3 and TAB4.

A more complex option is to split the same file or table among multiple Replicat processes. This might be necessary, for example, when one million inserts and updates per day might occur against FILE1, while in the rest of the system only 100,000 inserts and updates occur. In this case, the optimal configuration may be two Replicat processes for FILE1. This is accomplished in two steps:

1. Let Extract split the data into two trails. Each trail contains half the data for FILE1. To split the data, use the WHERE, RANGE, or FILTER clause of the Extract file parameter.
2. Assign a Replicat process to each of the resulting trails.

Splitting up tables among different Extract processes may temporarily upset original transaction integrity boundaries, since two or more processes may be replicating a single transaction.

The following Extract parameter file splits $DATA.MASTER.ACCOUNT into two trails.

```
EXTRACT DEMO
EXTTRAIL \NY.$DATA1.GGSDAT.E1
TABLE $DATA.MASTER.ACCOUNT, WHERE (ACCOUNT < 500000);
EXTTRAIL \NY.$DATA3.GGSDAT.E2
TABLE $DATA.MASTER.ACCOUNT, WHERE (ACCOUNT >= 500000);
```

A Replicat group is then dedicated to process each of the trails above.

Potential problems with audit compressed files

When replicating records selected with WHERE criteria from a source file with audit compression, update records can be missed (deletes and inserts will always be extracted). You can guarantee that all updates are processed by omitting fields that are not part of the primary key from your WHERE clauses. Primary key fields are always present in compressed update records.

When mapping selected columns with COLMAP, audit compression also causes potential conflicts. If the key of the target file includes a field not contained in the key of the source, target updates can fail. Updates require the presence of the entire key to guarantee success.

The easiest method for avoiding these conflicts is to turn off audit compression for source tables and files. This may or may not be feasible depending on the characteristics of your transaction load.

Conflicts with updating the target

If both GoldenGate and another application are allowed to update a target, conflicts can arise unless you establish rules to avoid conflicts. For example, application #1 might update a record in the source database that application #2 has deleted from the target database. In such cases, it is impossible for GoldenGate to apply the update on the source database at the target because the record to update no longer exists.

As a general rule, Replicat should have control over ranges of data that other applications cannot update. However, if conflicts are tolerable, GoldenGate provides features that allow operations to continue uninterrupted when errors occur:

- Use the REPERROR (<error>, IGNORE) parameter entries to ignore errors that normally cause transactions to abort.
- Use OVERRIDEDUPS and INSERTMISSINGUPDATES to ensure all updates are inserted.
- Review the Replicat discard file for operations that failed, and determine corrective measures.

Many-to-one replication

When replicating many files to one file (collecting), applications should ensure that each source file manages a specific range of keys. If different source files can update the same key value, there can be conflicts at the target. For example, if two source tables receive an insert with the same key, both operations cannot be applied at the target because a duplicate error will result (Guardian error 10, SQL error -8227).

GoldenGate provides several alternatives for dealing with this problem. One is the HANDLECOLLISIONS parameter that directs Replicat to insert the latest version of the record, even if the key exists. HANDLECOLLISIONS ignores missing update and delete conditions. Another option is to restrict the range of values replicated from each source by means of WHERE criteria. Most often the best alternative is to avoid the possibility of such conflicts as part of the application's processing rules.

Bi-directional replication

Sometimes, you may want to have two or more files replicating data to each other. In such cases, have each file manage a unique range of keys directly, as in the many-to-one case above. The difference here is that each file will hold data it manages, along with data replicated from the other file. In this way, each file can act as a backup for the other. The application should ensure that replicated data is read-only in such cases.

Since both files must be replicated, each replicated change will itself be extracted and replicated back to its source, which will cause errors. There are two methods for avoiding this condition:

- Restrict the ranges of key values that are extracted and replicated using WHERE criteria.
- Use the IGNOREREPLICATE parameter in Extract processing. This parameter causes Extract to discard any operations that were applied by Replicat processes.

Replicating data to non-TMF enabled databases

You can stimulate overall system performance by implementing buffering on your non-TMF Enscribe databases. To do so, turn on file buffering for target database files with the FUP ALTER <file>, BUFFERED command. This imposes no real risk since the data is mirrored at the source system and can be recovered from there.

Use the NOAUDITREPS Replicat parameter to avoid unnecessary event messages regarding non-audited target files.

Replicating new SQL columns

To replicate new SQL columns that were created since the current Extract and Replicat processes were started, include REPNEWCOLUMNS in the Replicat parameter file. REPNEWCOLUMNS replicates the SQL ALTER TABLE ADD COLUMN statements to create the new columns in the target.

Alternatively, you can specify GETNEWCOLUMNS to update table definitions when a column change is detected on a source table. GETNEWCOLUMNS ensures that data in columns created after Replicat startup (using ALTER TABLE ADD COLUMN on the source system) are accounted for.

Configuring for maximum throughput

You can maximize throughput by modifying Extract, Replicat, or both. This section details strategies for implementing GoldenGate parameters to achieve data management that suits your needs.

Extraction

Techniques for maximizing throughput on Extract depends on whether the source system produces TMF trails or non-TMF logs.

TMF extraction

In most cases, only a single instance of Extract is required to extract and transmit data to the target system. A single Extract is advantageous because TMF audit trails are only read once.

In rare cases, extracting high volumes of SQL UPDATE statements requires multiple instances of Extract.

Non-TMF data extraction

Non-TMF logging is linearly scalable by adding more Logger processes to the configuration. Since there is no penalty for adding Logger processes to the configuration, GoldenGate recommends allocating plenty of slack for high volume activity. In most cases, two or three Logger processes is more than enough to achieve the desired throughput.

Replication

To achieve required throughput, more Replicat processes may be required. This is because Replicat's I/O activity tends to be random access, as opposed to Logger and Extract I/O, which is serial, blocked and buffered.

You can add Replicat processes to achieve near linear performance gains. However, to ensure good performance, no more than three Replicat processes should read each GoldenGate trail. Otherwise, excessively redundant reads result, sometimes causing contention issues on the trail's disk.

Latency issues

Latency often refers to the difference in time between when an update occurs on the source database and when that same update is replicated on the target database. In this respect, latency measures the amount of time "behind" that the target system is from the source system, and can be important when determining the target database's accuracy. Database latency is especially important in certain bi-directional scenarios when two systems might update the same record in different databases at virtually the same time.

Another measure of latency is the lag between an update on the source and the time at which that update has been stored on the target system for later replication. This measure of latency represents the potential for the amount of data lost in the event of the disaster. Once data has been transmitted to the target, it will be replicated eventually and is not exposed to the risk of disaster.

Capacity planning

Through testing, GoldenGate has compiled some capacity planning guidelines, presented in the following sections. Consider these statistics as guidelines; actual performance depends on many of the factors previously discussed including network topology, operating systems, etc.

TMF data extraction

GoldenGate Software has observed TMF audit scanning rates of over ten gigabytes per hour on an S7000 system. A single Extract process has been observed to output over three gigabytes per hour. In this case, the output figures are far less than the audit generated, since extracted data does not include alternate keys, SQL indexes, FUP RELOAD information and assorted audit records.

Non-TMF data extraction

Non-TMF extracts are linearly scalable. Therefore, the potential extraction rate of data is close to the system limits for existing application activity. GoldenGate Software has observed I/O extraction rates on a two-processor, eight-disk S7000 system above 400 I/Os per second per Logger process, which can scale on larger systems well above 2000 I/Os per second using multiple Logger processes.

Data transfer into GoldenGate trails

The potential for data transfer is around 75-80% of the communication link's actual potential. When this limit is reached, you can split data into multiple trails to achieve greater throughput with parallelism.

Replicat throughput

The potential throughput of Replicat is greater than that of the database I/O performed on the source system. Replicat performs essentially the same I/Os on the target system as were performed on the source system, excluding reads. In addition, Replicat uses transaction grouping features as mentioned earlier to improve TMF-related performance

Changing default component names

GGSCI provides default names for processes, parameter files, and report files. You may wish to change these defaults to make them more descriptive. For example, you may wish to denote the parameter files and reports associated with a particular Extract or Replicat group (remember, you can have multiple Extracts and Replicats).

To change default component names:

1. Launch GGSCI.
2. Specify the define =GGS_PREFIX using the following syntax.

   ```
   GGSCI> ADD DEFINE =GGS_PREFIX, CLASS MAP, FILE $<prefix>
   ```

   Where: <prefix> consists of two letters.

   Consider the example:

   ```
   GGSCI> ADD DEFINE =GGS_PREFIX, CLASS MAP, FILE $EF
   ```

   This example changes the following default components.

   ❍ The Manager process name changes from $GGMGR to $EFMGR.
   ❍ Logger process names become $EFLnn instead of $GGLnn.
   ❍ Parameter files are stored in the EFSPARM subvolume rather than GGSPARM.
   ❍ Report files are stored in the EFSRPT subvolume rather than GGSRPT.

- ❍ Extract processes are called $EFEnn rather than $GGEnn.
- ❍ Replicat processes are called $EFRnn rather than $GGRnn.
- ❍ SyncFile processes are called $EFSnn rather than $GGSnn.
- ❍ Coordinator processes are called $EFCnn rather than $GGCnn.

3. One way to tell GGSCI and application programs bound with GGSLIB where to establish and retrieve configuration information is to use the =GGS_AUDCFG define. Alternately the location can be specified when running BUILDMAC or NLDLIB. When this location is not provided with one of these methods, the default is $SYSTEM.GGS.AUDCFG.

```
GGSCI> ADD DEFINE =GGS_AUDCFG, CLASS MAP, FILE <config file>
```

Where: <config file> is a file name, and the file portion of the file name is no longer than six characters.

Using wildcards

You can use wildcard arguments to express volumes, subvolumes, files and tables. GoldenGate allows wildcards to be expressed as a question mark (?) or an asterisk (*). An asterisk matches any number of characters, whereas a question mark matches only a single character.

Example For this example, the wildcard expression refers to any file set in the specified volume and subvolume.

```
FILE $DATA1.MYSUB.*;
```

Example In this next example, the wildcard expression refers to any volume $DATAn, where n represents the fifth character in the volume name, and any file in the specified subvolume.

```
FILE $DATA?.MYSUB.*;
```

By default, GoldenGate initially allocates 100 wildcard entries. You can change this initial allocation using the MAXWILDCARDENTRIES parameter in the GLOBALS, Extract, and Replicat parameter files. Once this initial MAXWILDCARDENTRIES allocation is exhausted, the program will allocate an additional 100 entries each time it needs more.

When you specify MAXWILDCARDENTRIES in the GLOBALS parameter file, that specification becomes the default. You can override that default using the MAXWILDCARDENTRIES parameter in the Extract or Replicat parameter files.

Most parameters that specify file names or table names can use wildcard expressions. Exceptions are documented in the parameter's description.

CHAPTER 4
Configuring Initial Data Synchronization

Before running GoldenGate for the first time you must synchronize the source and target databases. This chapter addresses initial data synchronization when the source is TMF-enabled. Initial load for non-TMF is performed via a trail; it acts as any other change extract.

Initial data synchronization

There are many ways you can set up your initial data load. The method you choose depends, in part, on the database types of your source and target. For example, if you are replicating from a NonStop source to a NonStop target, you may queue your data in trails, load directly from the source to the target (called bulk loading), etc. If you are replicating from a NonStop source to an Oracle target, your choices expand to include Direct Bulk Load, which interacts with SQL*Loader. Other methods include:

- Backup and Restore
- Load to target using FUP or SQL
- Hybrid solutions using part GoldenGate, part operating system or RDBMS utilities

Regardless of the method chosen, the initial data synchronization should be run after initiating change capture and before configuring delivery. The steps are:

1. Configure and run Extract to capture all active database changes to a GoldenGate trail.
2. Perform initial load.
3. Configure and run Replicat.

Example steps for initial data load

This example shows how you might follow these three steps to configure change capture and delivery and perform an initial data load.

Configure and run Extract

You can configure this Extract process to read change records from a TMF audit trail, log trail, or flat file, and process them to a GoldenGate trail. This step is necessary before doing the initial load because it extracts and delivers ongoing changes to the source database, preserving data integrity for your business operations.

Instructions for configuring and running Extract can be found in "Configuring GoldenGate" on page 39.

Perform initial load using queuing method

You can perform your initial load using any of the methods; however this example addresses how to use GoldenGate to do the initial load by queuing data to a GoldenGate trail.

1. Create one Extract parameter file to read directly from each source database.
2. Use the NonStop text editor to set up an Extract parameter file to include the following information.
    - The SOURCEISFILE parameter indicates that data should be retrieved directly from the table.
    - The format for the target file, usually FORMATASCII (for example, FORMATASCII, SQLLOADER or FORMATASCII, BCP).
    - If you are transmitting data to a system other than NonStop, or to a NonStop system over TCP/IP, include the name of the remote TCP/IP host and port number of the remote Collector.
    - The name of the local output file (EXTFILE) or the remote file (RMTFILE) to which the Extract program writes extract information. If you need to write to a series of trails, then add the MAXFILES 2 to the remote trail's parameter file. MAXFILES will append a six-digit trail sequence to the remote trail's file name.
    - The name of the file or table to extract (FILE or TABLE parameter).
    - Other optional parameters, including clauses for selecting records, column mapping, or data conversion.
3. Configure Replicat as a batch task, specifying the SPECIALRUN and BEGIN and END parameters.
4. Start the initial data load:

```
TACL> RUN EXTRACT /IN <parameter file>/
TACL> RUN REPLICAT /IN <parameter file>/
```

Configure and run Replicat

When your initial data load finishes writing to its trails, configure Replicat on the target system. This can be the same parameter file you use for ongoing Replicat work, however, you will need to add HANDLECOLLISIONS and END parameters, run the batch, then remove those parameters before beginning ongoing change extract.

1. Configure Replicat on the target system, including HANDLECOLLISIONS and END in the parameter file. The END parameter is the time recorded for the completion of the Extract process.
2. Start Replicat with the START REPLICAT command.
3. When Replicat stops, remove the HANDLECOLLISIONS and END parameters.
4. Start Replicat for incremental data synchronization.

Direct load

Using direct load, you can extract data directly from the source tables and send it, in a large block, to Replicat. You may do this on any operating system and database combination GoldenGate supports (e.g. NonStop to NonStop, NonStop to Oracle, Oracle to NonStop).

To run direct load:

1. Define an Extract group:

   ```
   GGSCI> ADD EXTRACT <group name>, SOURCEISFILE
   ```

2. Define a Replicat group:

   ```
   GGSCI> ADD REPLICAT <group name>, SPECIALRUN
   ```

   Replicat is automatically started by Manager, at Extract's request.

3. Create the Extract parameter file.

For the Extract parameter file:

Figure 10 Sample Extract parameter file

```
EXTRACT INITEXT
RMTHOST targethost, MGRPORT 7809
RMTTASK REPLICAT, GROUP INITREP
TABLE $DATA.MASTER.ACCOUNT, AUTOTRUNCATE;
TABLE $DATA.MASTER.PRODUCT, AUTOTRUNCATE;
TABLE $DATA.MASTER.CUSTOMER AUTOTRUNCATE;
```

- AUTOTRUNCATE sends a PURGEDATA command to Replicat before any data is processed. This ensures the target is clean and ready to receive data.

  NOTE Use AUTOTRUNCATE with extreme caution, as it causes all existing data to be purged from the target file. Refer to "Purging records for an initial load" on page 168 of the *Reference Guide* for more information.

- RMTHOST establishes the remote TCP/IP host and port number.
- RMTTASK instructs Manager on the target system to start Replicat with the specified GROUP name.
- The TABLE parameters identify the source tables.
- Specify SOURCEISFILE in the parameter if you wish to include a SOURCEISFILE option:
  - SELECTVIEW: Selects data from a specified SQL view in the file parameter. Without SELECTVIEW, Extract selects data from the base table of the view, then maps the base table columns to the view columns (this also occurs when processing audit trails and a view is specified).
  - FASTUNLOAD: Processes the file or table several times faster than the default method. Records are output in random order, rather than primary key order. FASTUNLOAD has no effect when an SQL view is specified. The file parameter option PARTITIONS can restrict the data retrieved to a certain subset of the file or table
  - FASTUNLOADSHARED: Allows a shared open of the source file or table. Use this only on files that are not receiving updates at the same time data is being extracted.

For the Replicat parameter file:

Figure 11 Sample Replicat parameter file

```
REPLICAT INITREP
USERID GoldenUser, PASSWORD pass
SOURCEDEFS $DATA.DIRDEF.SRCDEF
MAP $DATA.MASTER.ACCOUNT, TARGET $DATA3.MASTER.ACCOUNT;
MAP $DATA.MASTER.PRODUCT, TARGET $DATA3.MASTER.PRODUCT;
MAP $DATA.MASTER.CUSTOMER, TARGET $DATA3.MASTER.CUSTOMER;
```

- USERID and PASSWORD are required to access the target database.
- SOURCEDEFS identifies the file containing the source data definitions.
- The MAP parameters map the source tables to the target tables, based on the data definitions in SOURCEDEFS.

4. Start Extract:

```
GGSCI> START EXTRACT INITEXT
```

Using wildcards

Wildcards can be used for the FILE and TABLE statements in direct load parameter files, but not for views.

Refer back to the example of an Extract group added with the SOURCEISFILE parameter in "To run direct load:" on page 59. If the ACCOUNT, PRODUCT and CUSTOMER files are the only files on $DATA.MASTER, the Extract parameters could be changed to use wildcards as shown below.

Figure 12 Direct load parameter file with wildcards

```
EXTRACT INITEXT
RMTHOST targethost, MGRPORT 7809
RMTTASK REPLICAT, GROUP INITREP
TABLE $DATA.MASTER.*, AUTOTRUNCATE;
```

Direct bulk Load

If you are loading to an Oracle target, you may choose to use direct bulk load. Direct bulk load is the fastest technique for capturing and delivering data to SQL*Loaders. Extract sends the data, in a large block, to Replicat. Manager dynamically starts Replicat, which communicates directly with SQL*Loader using an API.

NOTE You can only use this direct bulk load from NonStop to Oracle.

To run direct bulk load:

1. Define an Extract group:

```
GGSCI> ADD EXTRACT <group name>, SOURCEISFILE
```

2. Define a Replicat group:

```
GGSCI> ADD REPLICAT <group name>, SPECIALRUN
```

Replicat is automatically started by Manager, at Extract's request.

3. Create the Extract and Replicat parameter files.

Sample direct bulk load parameter files

Figure 13 Sample Extract parameter files

```
EXTRACT INITEXT
RMTHOST targethost, MGRPORT 7809
RMTTASK REPLICAT, GROUP INITREP
TABLE $DATA.MASTER.ACCOUNT;
TABLE $DATA.MASTER.PRODUCT;
TABLE $DATA.MASTER.CUSTOMER;
```

For the Extract parameter file:

- RMTHOST establishes the remote TCP/IP host and port number.
- RMTTASK instructs Manager on the target system to start Replicat with the specified group name.
- The TABLE parameters identify the source tables.

Figure 14 Sample Replicat parameter file

```
REPLICAT INITREP
USERID GoldenUser, PASSWORD pass
BULKLOAD
SOURCEDEFS /GGS/DIRDEF/SRCDEF
MAP $DATA.MASTER.ACCOUNT, TARGET master.account;
MAP $DATA.MASTER.PRODUCT, TARGET master.product;
MAP $DATA.MASTER.CUSTOMER, TARGET master.customer;
```

For the Replicat parameter file:

- USERID and PASSWORD are required to access the target database.
- BULKLOAD tells Replicat that SQL*Loader will load the target tables.
- SOURCEDEFS identifies the file containing the source data definitions.
- The MAP parameters map the source tables to the target tables, based on the data definitions in SOURCEDEFS.

4. Start Extract:

```
GGSCI> START EXTRACT <group name>
```

Synchronizing NonStop databases through TCP/IP

You can synchronize two NonStop tables or files on different systems over a TCP/IP connection. Use the following steps:

1. Start the Collector on the target system.
2. Create a parameter file to perform initial file extraction over the TCP/IP link.

Figure 15 Sample Extract parameter file

```
SOURCEISFILE, FASTUNLOAD
FORMATLOAD
RMTHOST 192.23.39.12, PORT 7829
RMTFILE $D3.INIDAT.TRANSTAB, PURGE
FILE \SRC.$D2.MYDB.TRANSTAB;
```

For this example parameter file, named GGSPARM.TRANINI, you are identifying the Collector on the remote host.

- SOURCEISFILE, FASTUNLOAD directs Extract to retrieve data directly from the blocks of the table.
- FORMATLOAD directs Extract to format the data compatible with FUP or SQLCI LOAD.
- RMTHOST identifies the IP address and port of the Collector process. When NonStop is the receiver, a separate Collector is required for each simultaneously active session.
- RMTFILE identifies a flat file that holds the extracted data until the load is finished.
- FILE identifies the source file or table from which to extract the data.

3. Run Extract to extract the data into a flat file on the target system.
4. Use FUP or SQLCI to insert the data into the target system, similar to:

```
SQLCI> LOAD $D3.INIDAT.TRANSTAB, $D4.MYDB.TRANSTAB, RECIN 236, RECOUT
236;
```

   The figures for RECIN and RECOUT are derived from Extract's output in $S.#TRAN, which includes the physical length of the records in the target. For Enscribe, this is the same as the record length returned by FUP INFO. For SQL, the size will vary and can be returned from RECORDSIZE column in the FILE table from the source table's catalog.

Controlling the IP process for Replicat

Although you can configure multiple initial load Extracts and Replicats, by default the Replicats will inherit the IP process of the Manager running on the target. This results in a single IP channel that does not spread the load across the processors.

To configure the Extract and Replicat pairs to use different channels, you can use static Replicats as shown in the example below.

1. Configure multiple Extracts to use the PORT parameter, rather than MGRPORT. Assign a different port to each.

```
EXTRACT extl1
RMTHOST 192.168.118.145, PORT 12345
RMTTASK REPLICAT, GROUP repl1
EXTRACT extl2
RMTHOST 192.168.118.145, PORT 12346
RMTTASK REPLICAT, GROUP repl2
```

2. Start static Replicats using the runtime parameter INITIALDATALOAD with the -p option to assign the port from the parameter file.

```
TACL> ASSIGN STDERR, $0
TACL> ADD DEFINE =TCPIP^PROCESS^NAME, FILES $ZTC1
TACL> RUN REPLICAT/IN GGSPARM.REPL1,OUT GGSRPT.REPL1/INITIALDATALOAD -p
12345
TACL> ASSIGN STDERR, $0
TACL> ADD DEFINE =TCPIP^PROCESS^NAME, FILES $ZTC2
TACL> RUN REPLICAT/IN GGSPARM.REPL2,OUT GGSRPT.REPL2/INITIALDATALOAD -p
12346
```

NOTE Since the Replicats are started statically, they will not be restarted by Manager if there is a system problem.

Loading Oracle, Microsoft or Sybase SQL Server tables

NonStop tables and files can be synchronized with Oracle or SQL Server tables in very much the same way as NonStop-to-NonStop synchronization.

To load to Oracle or SQL Server:

1. Run DEFGEN to export source data definitions to the target system. Be sure to satisfy any other prerequisites.
2. Start the Collector on the target system:

   For UNIX:

   ```
   $server –d /ggs/mydb.def 2> server.log &
   ```

   For Windows:

   ```
   server –d \ggs\mydb.def 2> server.log
   ```

   Use the –d option to specify the definitions file created by DEFGEN (mydb.def).
3. Create an Extract parameter file to perform initial table extract over the TCP/IP link.
4. Run the Extract program to extract the data into a flat file on the target system:

   ```
   TACL> RUN GGS.EXTRACT /IN GGSPARM.TRANINI, OUT $S.#TRAN/
   ```

   This command creates a flat file on the target. If you specified the FORMATASCII, SQLLOADER in a parameter file for Oracle, GoldenGate generates the flat file in a format that SQL*Loader can read. If you specified FORMATASCII, BCP in the parameter file for SQL Server, GoldenGate generates a flat file that is compatible with the BCP utility.
5. Create a Replicat parameter file using the MAP parameter to map between source and target tables.
6. Run Replicat to create files called TRANSTAB.ctl and TRANSTAB.run for Oracle, and TRANSTAB.bat and TRANSTAB.fmt for SQL Server. These files contain mapping definitions and run commands required to load.

   For UNIX:

   ```
   replicat paramfile/ggs/dirprm/tranini.prm
   ```

For Windows:

```
C:\replicat paramfile\ggs\dirprm\tranini
```

7. Load the data.

   For UNIX:

   ```
   $TRANSTAB.run
   ```

   For Windows:

   ```
   TRANSTAB.bat
   ```

Initial sync parameter file examples

This sections contains:

- An Extract parameter file example for Oracle running on UNIX.
- A Replicat parameter file example for Oracle running on UNIX.
- An Extract parameter file example for SQL Server running on Windows.
- A Replicat parameter file example for SQL Server running on Windows.

Sample NonStop to Oracle Extract parameter file

Figure 16 Extract parameter file for GGSPARM.ORAINI

```
SOURCEISFILE, FASTUNLOAD
FORMATASCII, SQLLOADER
RMTHOST ntbox12, MGRPORT 7809, PARAMS "-d c:\ggs\dirdef\source.def"
RMTFILE TRANSTAB.dat, PURGE
FILE \SRC.$D2.MYDB.TRANSTAB;
```

- FORMATASCII, SQLLOADER specifies to format the data to be compatible with Oracle's SQL*Loader utility.
- RMTFILE identifies TRANSTAB.dat as the source table.

Sample Replicat parameter file for Oracle

Figure 17 Replicat parameter file for /ggs/dirprm/tranini.prm

```
GENLOADFILES
USERID ME, PASSWORD ORAPW
SOURCEDEFS /ggs/mydb.def
MAP $D2.MYDB.TRANSTAB, TARGET ORATRANS;
RMTFILE /ggsdat/tranini, PURGE
FILE \SRC.$D2.MYDB.TRANSTAB;
```

- GENLOADFILES generates load control files and then quits. These control files generate maps, even between dissimilar tables.
- USERID and PASSWORD specify the database log on.
- SOURCEDEFS specifies the location of the NonStop definitions exported by DEFGEN (these are required to generate a load map).

- MAP specifies the source to target relationship of the NonStop to Oracle table.
- Errors are output to the screen and detailed messages are output into the TRANSTAB.err and TRANSTAB.log files.

Sample Extract parameter file for SQL Server

Figure 18 Extract parameter file for GGSPARM.SQLINI

```
SOURCEISFILE, FASTUNLOAD
FORMATASCII, BCP
RMTHOST ntbox12, MGRPORT 7809, PARAMS "-d c:\ggs\dirdef\source.def"
RMTFILE C:\GGS\TRANSTAB.dat, PURGE
TABLE $DATA.MASTER.TRANS
```

- FORMATASCII, BCP specifies to format the data compatible with Microsoft's BCP utility.
- RMTFILE identifies TRANSTAB.dat as the source table. Using the .dat extension makes it compatible with the load functions.

Sample Replicat parameter file for SQL Server

To load data to SQL Server, you must use the BCP template provided by GoldenGate. You can call BCP from your Replicat parameter file or run it interactively from the operating system shell. The template tells Replicat how data is laid out in the SQL Server target.

Figure 19 Replicat parameter file for GGSPARM.TRANINI

```
GENLOADFILES BCPFMT.TPLTARGETDB
TARGETDB MYAPP, USERID MYNAME, PASSWORD MSPW
SOURCEDEFS c:\ggs\mydb.def
MAP $D2.MYDB.TRANSTAB, TARGET SCHEMA.ORATRANS;
```

Loading initial data from Windows and UNIX

Use Replicat to load data from a Windows or UNIX system into a NonStop target database. See the *GoldenGate for Windows and UNIX Administrator Guide* for details.

Integrating source and target data

When only a subset of source rows or columns are needed in the target, you can use one of the following techniques to integrate selected data into your target. These techniques include:

- Selecting on the source with WHERE or FILTER.
- Mapping columns on the target with COLMAP.

When the data source is a SQL table, you can specify SQL Views. This allows automatic filtering of columns before transmission. You can also specify a view that joins multiple tables, for denormalization purposes.

Data transformation (such as six-to-eight digit date conversion) takes a little extra effort during the load process. There are a couple of ways to achieve initial loads in this situation.

The first solution involves extracting the entire table into a flat file. In this case, do not specify FORMATASCII. Next use Replicat to load the table using the SPECIALRUN parameter. This method, while slower than native loads, is often sufficient and allows field conversion functions to be used during replication.

The second solution is to perform all data mapping on the NonStop before transmission on the target side. This means that all conversion work is performed by Extract. Using this strategy can result in less network traffic, since filtering can be performed before data reaches the pipe. However, this can also require the creation of a dummy table or DDL definition on the NonStop side that mimics the structure of the real target table.

Distributing extracted data

In addition to extracting and replicating database changes, Extract can forward and distribute changes that have already been extracted. This process is known as *data pumping*.

Use data pumping in the following scenarios:

- A network or target system may be down for an extended time, but extraction or logging activities must occur constantly.
- Data extracted by Logger needs to be forwarded over TCP/IP to non-NonStop systems.

Running Extract for these purposes is nearly identical to capturing data from TMF audit trails. To run Extract in this manner, perform the following tasks.

1. Using the EXTTRAILSOURCE or LOGTRAILSOURCE option, create an initial Extract checkpoint with the GGSCI ADD EXTRACT command.
2. Add a local or remote GoldenGate trail with the GGSCI ADD EXTTRAIL or ADD RMTTRAIL command. By adding the trails, you direct Extract where to write the data you need.
3. Set up an Extract parameter file.
4. Start Extract using the GGSCI START EXTRACT command.

Direct file extraction

Rather than capturing from trails, you can extract directly from a file or a sequence of files. You can read a file directly only when the following conditions are true:

- The file or sequence of files is entry-sequenced.
- Only inserts occur against the files (no updates).
- Records are inserted only at the end of the file.

Use this feature when:

- The method of logging is non-TMF.
- The files are BASE24 TLF or PTLF.
- The input files meet the conditions described above.
- You want to transfer the batch file contents a little at a time throughout the day (“trickle” transfer), rather than all at once at the end of the day.

To extract directly from a file:

1. Enter a GGSCI ADD EXTRACT command, specifying the FILETYPE parameter. FILETYPE indicates the type of file from which you are reading.
2. If more than one file in a sequence might be open at a time, start Extract for each file in use simultaneously. Enter an ALTINPUT parameter in each process's parameter file with a RANGE option to distribute the files among the processes. For further details, see the *Reference Guide*.

Batch processing

You can configure Extract and Replicat to run in batch when capturing and delivering incremental changes is not appropriate for the application. You can configure ongoing batch runs for a specific time daily, or special, one-time batch runs.

One-time database extraction

You can run Extract against a specified period of audit trail or GoldenGate trail data a single time. Do this, for example, to extract changes to a particular account in a database over the course of a day.

To extract changes for a specific period, perform the following steps.

1. Set up a parameter file using the NonStop editor.
2. Use SPECIALRUN to capture data from TMF-audit trails. SPECIALRUN indicates that no checkpoints are maintained.
3. To extract data from a GoldenGate trail, use the SPECIALRUN, EXTTRAILSOURCE or LOGTRAILSOURCE option.
4. Set up BEGIN and END parameters to designate the period of activity to extract.
5. Designate an EXTFILE or RMTFILE rather than an extract trail. If you require multiple trails, add the MAXFILE argument to the EXTFILE or RMTFILE parameter.
6. Specify additional parameters as needed.
7. Initiate Extract from TACL, as in this example.

```
TACL> RUN EXTRACT /IN GGSPARM.SPECEXT, OUT GGSRPT.SPECEXT/
```

Trickle batch processing

When you are extracting batch files using RMTBATCH, you may need to perform the following steps:

1. Use the SYSKEYCONVERT parameter in the Extract parameter file if the input record's length is variable. This specifies the format of the SYSKEY in the output.
2. Use the POSITIONFIRSTRECORD parameter to reread an input file when you have used SYSKEYCONVERT. POSITIONFIRSTRECORD resets Extract to read from the beginning of the input file.

Determining the next file

Use ALTINPUT for direct file extraction. With ACI files, multiple files can be in use at one time.

For example, processing can continue on Monday's file after midnight, while Tuesday's file is opened for new data. To handle a multiple file situation, run more than one Extract process for the file sequence. Use the ALTINPUT RANGE option to distribute the files across the processes so that Extract never processes two files in sequence. You can also use ALTINPUT to specify the access mode of the file open, and to move Extract to the next sequential file if an application has a file open that it is not updating.

By default, Extract looks for the next alphabetical file name. The file name must conform to a template for the file type, which defaults to predefined characteristics. You can also specify the template by parameter.

If the file type is ACITLF or ACIPTLF, the template is in the form $VOL.SUBVOL.XXYYMMDD, where XX is a two character prefix, followed by a six digit date.

If the file type is ACITLFX or ACIPTLFX, the template is in the form $VOL.SUBVOL.XMMDDNNN, where X is a one character prefix, followed by a month, day and three digit sequence number.

When specifying any of the ACI file types in the FILETYPE option, do not include the date or sequence number. The next file is the one following the current file in name order, and must also satisfy any RANGE criteria in the ALTINPUT parameter.

If the file type is ENTRY, you specify the template in the ALTINPUT parameter TEMPLATE option. NonStop wildcards are acceptable. For example, the template $DATA*.MYDAT.FL* processes files starting with FL residing on different $Data volumes.

When using FILETYPE ENTRY, specify the first file to process, not the file prefix. By default, the next file is the next file name to fit the template. As an alternative, you can use FILETYPE USENEXTMODIFIED. This option selects the next file modified after the current file that also fits the template.

When the next file is processed

Before moving to the next file in a sequence, Extract must process the entire contents of the current file. By default, Extract uses the following rules to determine that the current file has been exhausted and the next file is ready for processing.

- End-of-file was reached in current file at least five seconds earlier, and no new data has appeared since.
- No processes have the current file open for write access.
- The next file exists and has at least one record in it.
- The next file was modified after the current file.

You can modify these rules with the NOWAITNEXTMODIFIED, WAITNEXTRBA, and OPENTIMEOUT options for the ALTINPUT parameter.

CHAPTER 5
Configuring GoldenGate Security

This chapter helps you implement GoldenGate's security features to secure the GoldenGate environment and the data being processed.

Overview of security features

The following are ways to secure your GoldenGate environment and the data being processed:

Table 1 GoldenGate security features

Security feature	Description
Encryption	Options are available for encrypting and decrypting: ◆ data in an extract file or trail ◆ database passwords ◆ data sent across TCP/IP See "Using encryption" below.
Command security	Sets user-level permissions for accessing GoldenGate commands through GGSCI. See "Using command security" on page 73.

Using encryption

This section contains instructions for encrypting and decrypting the following:

- The trail or extract file that holds data being processed by GoldenGate. You can encrypt any local or remote trail or file.
- The database password used by the Extract and Replicat processes and other processes to log into the source and target databases. This user is specified in the associated parameter files with the USERID parameter. The PASSWORD argument of USERID can be supplied as plain text, or it can be encrypted using the steps in this topic.
- The data sent across TCP/IP. On the target system, GoldenGate decrypts the data before writing it to the GoldenGate trails (unless trail encryption also is specified). By default, data sent across a network is not encrypted.

The following encryption methods are used:

- To encrypt trail or extract files, GoldenGate uses 256-key byte substitution to encrypt the files. All records going into those files are encrypted both across any data links and within the files themselves.
- To encrypt the database password or data sent across TCP/IP, GoldenGate uses Blowfish encryption from Counterpane Internet Security. Blowfish is a symmetric block cipher that can be used as a drop-in replacement for DES or IDEA. GoldenGate's implementation of Blowfish can take a variable-length key from 32 bits to 128 bits. Blowfish encryption can be combined with GoldenGate trail encryption.

To encrypt trail or extract files

NOTE This feature cannot be used when FORMATASCII is used to write data to a file in ASCII format. The trail or file must be written in internal GoldenGate format.

1. In the Extract parameter file, include the following parameter.

```
ENCRYPTTRAIL
```

Data written to all extract files or trails listed in the parameter file after ENCRYPTTRAIL will be encrypted.

To disable encryption for any files or trails listed in the parameter file, precede their entries with the following parameter:

```
NOENCRYPTTRAIL
```

2. In the Replicat parameter file, include the following parameter so that Replicat decrypts the data for processing.

```
DECRYPTTRAIL
```

You also can use DECRYPTTRAIL for an Extract data pump to decrypt the data for column mapping, filtering, transformation, and so forth. You can then leave it decrypted for downstream trails or files, or you can use ENCRYPTTRAIL to encrypt the data again before it is written to those files.

To encrypt the database password

1. Run GGSCI and issue the ENCRYPT PASSWORD command to generate an encrypted password. The command provides the following options.
   - The default ENCRYPT PASSWORD command, without any arguments, generates an encrypted password using a default key that is randomly generated by GoldenGate.

     ```
     ENCRYPT PASSWORD <password>
     ```

   - ENCRYPT PASSWORD with the ENCRYPTKEY <keyname> generates an encrypted password using a user-defined key contained in the ENCKEYS lookup file.

     ```
     ENCRYPT PASSWORD <password> ENCRYPTKEY <keyname>
     ```

     For <keyname>, specify the logical name for the key you want to use, as it appears in the local ENCKEYS file. To use this option, you must first generate a key, create an

ENCKEYS file on the local system, and create an entry in the file for the generated key. For instructions, see "To generate encryption keys" on page 72.

2. The encrypted password is output to the screen when you run the ENCRYPT PASSWORD command. Copy the encrypted password and paste it into the USERID parameter in the parameter file as follows.

```
USERID "<user id>",
PASSWORD "<password>",
[ENCRYPTKEY {DEFAULT | <keyname>}]
```

Where:

- <user id> is the database user name for the GoldenGate process or (Oracle only) a host string.
- <password> is the encrypted password that is copied from the ENCRYPT PASSWORD command results.
- ENCRYPTKEY DEFAULT is required if the password was encrypted using ENCRYPT PASSWORD without the ENCRYPTKEY option.
- ENCRYPTKEY <keyname> is required if the password was encrypted using ENCRYPT PASSWORD with the ENCRYPTKEY <keyname> option. Specify the logical name of the key as it appears in the ENCKEYS lookup file.

To encrypt data sent across TCP/IP

1. On the source system, generate one or more encryption keys and create an ENCKEYS file. See "To generate encryption keys" on page 72.
2. Copy the finished ENCKEYS file to the GoldenGate directory on all target systems. The key names and values in the source ENCKEYS file must match those of the target ENCKEYS file, or else the data exchange will fail and Extract and Collector will abort with the following message:

```
GGS error 118 – TCP/IP Server with invalid data.
```

3. In the Extract parameter file, use the ENCRYPT option of the RMTHOST parameter to specify the type of encryption and the logical key name as shown:

```
RMTHOST <hostname>, MGRPORT <port>,
ENCRYPT BLOWFISH, KEYNAME <keyname>
```

Where:

- BLOWFISH specifies Blowfish encryption.
- <keyname> is the logical name for the encryption key you want to use, as it appears in the ENCKEYS file.

Example:

```
RMTHOST sys1, MGRPORT 7840, ENCRYPT BLOWFISH, KEYNAME superkey
```

4. If using a static Collector and Blowfish encryption, append the following additional parameters in the Collector startup string:

```
-KEYNAME <name>
-ENCRYPT <encrypt type>
```

Where:

- <name> is the name of the key.
- <encrypt type> is BLOWFISH.

Collector matches these parameters to those specified with the KEYNAME and ENCRYPT options of RMTHOST.

To generate encryption keys

You must create at least one encryption key and an ENCKEYS lookup file if you want to:

- Encrypt data sent across TCP/IP
- Use a *user-defined* key to encrypt the database password

This procedure is *not required* if you are using a default key to encrypt the database password or if you are encrypting a trail or extract file.

1. Create an encryption key value either by defining a literal string or by running GoldenGate's KEYGEN utility to create a key randomly. Key values can be a string of one to 24 alphanumeric characters without spaces, up to 128 bits (16 bytes) in length. Hex keys must begin with 0x. Literal keys must be enclosed within quotes, for example "Dailykey".

To use KEYGEN to generate a key, change directories to the GoldenGate home directory on the source system, and issue the following command. You can create multiple keys, if needed. The key values are returned to your screen.

```
TACL> RUN KEYGEN <key length> <n>
```

Where:

- <key length> is the encryption key length, up to 128 bits.
- <n> represents the number of keys to generate.

Example:

```
TACL> RUN KEYGEN 128 4
```

2. On the source system, open a new ASCII text file.
3. For each key that you generated, enter a logical name followed by the key value itself. Place multiple key definitions on separate lines. Do not enclose a key name or value within quotes; otherwise it will be interpreted as text. Use the following sample ENCKEYS file as a guide.

```
## Encryption keys

## Key name      Key value
superkey         0x420E61BE7002D63560929CCA17A4E1FB
secretkey        0x027742185BBF232D7C664A5E1A76B040
superkey1        0x42DACD1B0E94539763C6699D3AE8E200
superkey2        0x0343AD757A50A08E7F9A17313DBAB045
superkey3        0x43AC8DCE660CED861B6DC4C6408C7E8A
```

4. Save the file as ENCKEYS without an extension in the GoldenGate home directory. The name must be in upper case.

5. Copy the ENCKEYS file to the target GoldenGate installation directory. The key names and values in the source ENCKEYS file must match those of the target ENCKEYS file, or else the data exchange will fail and Extract and Collector will abort with the following message:

```
GGS error 118 – TCP/IP Server with invalid data.
```

Using command security

You can establish command security for GoldenGate to control which users have access to which GoldenGate functions. For example, you can allow certain users to issue INFO and STATUS commands, while preventing their use of START and STOP commands. Security levels are defined by the operating system's user groups.

To implement security for GoldenGate commands, you create a CMDSEC file in the GoldenGate directory. Without this file, access to all GoldenGate commands is granted to all users.

To implement command security

1. Open a new ASCII text file.
2. Referring to the following syntax and the example on page 74, create one or more security rules for each command that you want to restrict, one rule per line. Order the rules from the most specific (those with no wildcards) to the least specific. Security rules are processed from the top of the CMDSEC file downward. The first rule satisfied is the one that determines whether or not access is allowed.

   Separate each of the following components with spaces or tabs.

   ```
   <command name> <command object> <user group> <user> <YES | NO>
   ```

   Where:

   - <command name> is a GGSCI command name or a wildcard, for example START or STOP or *. Command names are not validated for accuracy.
   - <command object> is any GGSCI command object or a wildcard, for example EXTRACT or REPLICAT or MANAGER. Command objects are not validated for accuracy.
   - <user group> is the numeric ID of the Guardian user group, such as 100 or 255. You can use a wildcard to specify all groups.
   - <user> is the Guardian user numeric ID, such as 2 or 255. You can use a wildcard to specify all users.
   - <YES | NO> specifies whether access to the command is granted or prohibited.

3. Save the file as CMDSEC in the GoldenGate home directory.

CMDSEC examples

The following example illustrates the correct implementation of a CMDSEC file on a UNIX system.

Table 2 Sample CMDSEC file with explanations

File Contents	Explanation
`--GG command security`	Comment line
`STATUS REPLICAT 100 15 NO`	STATUS REPLICAT is denied to user 15 of group 100.
`STATUS * 100 * YES`	Except for the preceding rule, all users in 100 are granted all STATUS commands.
`START REPLICAT 255 * YES`	START REPLICAT is granted to all members of the Super (255) group.
`START REPLICAT * * NO`	Except for the preceding rule, START REPLICAT is denied to all users.
`* EXTRACT 200 * NO`	All EXTRACT commands are denied to all groups with ID of 200.
`* * 255 255 YES`	Grants the Super.Super user any command.
`* * * * NO`	Denies all commands to all users. This line covers security for any other users that were not explicitly granted or denied access by preceding rules. Without it, all commands would be granted to all users except for preceding explicit grants or denials.

The following *incorrect* example illustrates what to avoid when creating a CMDSEC file.

Table 3 Incorrect CMDSEC entries

File Contents	Description
`STATUS REPLICAT 100 15 NO`	STATUS REPLICAT is denied to user 15 of group 100.
`STOP * 100 * NO`	All STOP commands are denied to everyone in group 100.
`STOP * * 15 YES`	All STOP commands are granted to user 15.

The order of the entries in Table 3 causes a logical error. From the first rule (line 1), you can see that user 15 is a member of group 100. The second rule (line 2) denies all STOP commands to all members of group 100. The third rule (line 3) grants all STOP commands to user 15. However, because 15 is a member of the 100 group, he has been denied access to all STOP commands by the second rule.

The proper way to configure this security rule is to set the user-specific rule before the more general rule(s). Thus, to correct the error, you would reverse the order of the two STOP rules.

Securing the CMDSEC file

Because the CMDSEC file is a source of security, it must be secured. You can grant read access as needed, but GoldenGate recommends restricting write and delete access to everyone but the GoldenGate administrator. For example, a proper security string might be “NUUU”.

CHAPTER 6

Configuring the Manager and Collector

Two GoldenGate components facilitate day-to-day data management: the Manager and the Collector. This chapter introduces procedures and techniques for running both components.

Introducing Manager

Manager runs as a NonStop process, ensuring that GoldenGate components run and restart if a CPU failure or operator error occurs. Manager spawns a copy of itself so that tasks that take longer, such as duplicating a TMF audit trail, do not interfere with real-time tasks. Tasks are divided between the Manager and its child process accordingly. Manager tasks on NonStop, include:

- Starting Logger, Extract, Replicat and Syncfile.
- Monitoring and reporting status of GoldenGate processing.
- Starting the dynamic collector process on the target.
- Automatically restarting critical processes.
- Threshold reporting, such as when Extract falls behind the TMF-audit trail.
- Managing resources for the TMF audit trail, such as maintaining copies of audit trails on backup volumes.
- Purging trails when Extract and Replicat has finished with them.
- Pre-allocating log trail space for Logger processing.

Configuring and starting Manager

Now that you have your GGSCI prompt, you are ready to configure and start Manager. To configure Manager, create an appropriate parameter file. To control a GoldenGate Manager process, use the following commands.

Command	Description
`START MANAGER`	Starts Manager.
`STOP MANAGER`	Stops Manager. You can stop Manager gracefully, or forcefully with the ! option.
`REFRESH MANAGER`	Refreshes the Manager parameter file if you have made changes to it.

Creating and configuring the Manager parameter file

Enter Manager parameters in the GGSPARM.MGRPARM file, on the home volume. If no MGRPARM file exists, default management parameters are taken. To add parameters, edit this file using the EDIT PARAMS MGRPARM command.

Manager retrieves parameters as established by GGSCI ATCONFIG commands. These parameters affect audit trail resource management.

See the *Reference Guide* for more information about Manager parameters.

A Sample Manager parameter file

A Manager parameter file would be similar to this sample.

Figure 20 Sample Manager parameter file

```
TCPIPPROCESSNAME $ZTC2
PORT 7844
DYNAMICPORTLIST 7850 - 7880, 7895
CHECKMINUTES 30
PURGEOLDEXTRACTS $DATA1.GGSDAT.*, USECHECKPOINTS, MINKEEPDAYS 2
THRESHOLD 30
LAGREPORTMINUTES 60
LAGINFOMINUTES 10
LAGCRITICALMINUTES 10
LOGFILESBEHIND 2
LOGFILESBEHINDINFO 10
DOWNCRITICAL
DOWNREPORTHOURS 1
```

In this sample Manager parameter file

- TCPIPPROCESSNAME specifies the TCP/IP process. The default process is $ZTC0. Use the TCPIPPROCESSNAME parameter to specify a process other than the default.
- Specify the PORT parameter so Manager can create a dynamic Collector process.
- DYNAMICPORTLIST specifies up to 256 entries for ports to be dynamically assigned to processes started by Manager. If no dynamic ports are specified, Manager will start with 7840 and increment until it finds an available port.
- CHECKMINUTES 30 directs Manager to perform maintenance activities every 30 minutes. The default is 10 minutes.
- Use the PURGEOLDEXTRACTS parameter when multiple Replicat processes are reading a set of trails. For this sample, PURGEOLDEXTRACTS directs manager to purge old files from the trail $DATA1.GGSDAT.*. The options:
  - USECHECKPOINTS specifies that Replicat checkpoints are to be used to determine when Replicat has finished processing.
  - MINKEEPDAYS 2 purges files only after they have been closed for 2 days.
- THRESHOLD 30 directs Manager to generate an event message when the number of audit files remaining to be processed falls below 30%.
- LAGREPORTMINUTES 60 specifies that Manager check lag every 60 minutes.
- LAGINFOMINUTES 10 specifies that Manager report lag information to the event log every 10 minutes.

- LAGCRITICALMINUTES 10 specifies that Manager write a critical message to the event log when there is 10 minute lag.
- LOGFILESBEHIND 2 sends a critical message whenever a process lags a specified number of files behind the current log trail file.
- LOGFILESBEHINDINFO 10 sends an informational message whenever the process falls the specified number of files behind.
- DOWNCRITICAL sends a critical message whenever Extract or Replicat abends.
- DOWNREPORTHOURS sends reports of Extract and Replicat abending.

Starting, stopping, and refreshing Manager

You must start Manager before you can configure and run other GoldenGate components. The following example starts Manager in CPU 3. Manager selects a CPU in which to run a backup process for fault-tolerance.

```
GGSCI> START Manager, CPU 3, PRI 170
```

If Manager encounters a TCP/IP error, for example if it attempts to bind to a port that is in use, it retries the error every 60 seconds and does not abend after a set number of attempts. Unlike other GoldenGate processes, it does not use the TCPERRS file to set the delay and the number of retries.

Manager runs indefinitely, or until you enter the GGSCI STOP MANAGER command. One reason you would stop Manager is if you need to stop the Extract and Replicat groups it manages.

Changing a Manager parameter, however, does not require you to stop. Simply issue the REFRESH command; this forces Manager to re-read its parameter file and adjust its processing behavior accordingly.

See the *Reference Guide* for more information about GGSCI commands for Manager.

Configuring and running the Collector

The Collector collects data from Extract and writes data to files on the target system. Extract requests Manager to start a collector process when it sees data must transmit over TCP/IP to a remote trail. Once started, the Collector waits for and executes requests to write, open, and close files in the GoldenGate trail during Extract processing.

NOTE You do not need to run collector if data transmits over an Expand connection.

Configuring Collector

To configure a Collector, you must know the port the Collector will use, the host name or IP address where the remote trail resides, and edit your Manager and Extract parameter files. You may also specify a variety of operating options, described in the “Configuration examples” on page 78.

To configure and start Collector:

1. In the Manager parameter file, specify the port parameter, such as: PORT 7809.
2. In the Extract parameter file, specify the remote host parameter as follows:

```
RMTHOST <host>, [MGRPORT <port num>] [, <options>] [, ...]
```

Argument	Description
`<host>`	Either a remote host name or an IP address, such as: RMTHOST eastnode or RMTHOST 192.33.56.205.
`MGRPORT <port num>`	Specify the port that is defined in the Manager parameter file.
`<options>`	You can specify a variety of options, including Collector parameters. See "Creating and configuring the Manager parameter file" on page 76. See the *Reference Guide* for information about other RMTHOST options.

3. The remote host must be the same system on which Collector was started, and the port number must match the port number in the Collector startup command. See "Configuration examples" on page 78 for more information.
4. If you specify a remote host name in the remote host parameter, you must also enter the remote host name in the TCP/IP hosts file on the target system, or on the names server for your network. For example, if you specify: RMTHOST eastnode, you must make an entry similar to: 192.33.56.205 eastnode in the HOSTS file.

   If you specify the remote host IP address in the RMTHOST parameter, there is no need to make a corresponding HOSTS file entry.

Configuration examples

To configure Collector for port 5432 on remote host named eastnode:

1. In the Manager parameter file, specify: PORT 5432
2. In the Extract parameter file, specify: RMTHOST eastnode, MGRPORT 5432, and options if desired.
3. In the TCP/IP HOSTS file, enter: 192.33.56.205 eastnode

To configure Collector for the default port on remote host address 192.33.56.205:

1. In the Manager parameter file, specify: PORT 7809
2. In the Extract parameter file, specify: RMTHOST 192.33.56.205, and options if desired.
3. No TCP/IP hosts file entry is required.

The TCP/IP port

There are two ways to use Collector and your TCP/IP port: dynamically and explicitly. The dynamic method lets Extract request Manager to start Collector as needed. However, a user can explicitly start the Collector and let it run in the background, ready to transmit data as needed. This method is called the *explicit method*.

Dynamic method

The *dynamic method* is the default way to use Collector. The examples above illustrate how this is configured: a port is specified in the Manager parameter file, a remote trail is specified in the Extract parameter file, and, if required, the IP address is added to your hosts file.

Explicit method

When capturing data over TCP/IP to remote systems that do not support dynamic collectors, you must explicitly start a Collector on the target system before starting Extract. Each Extract must explicitly name the port to which it is writing, using the remote host parameter.

To explicitly configure your Collector, start GGSCI and enter the following:

```
TACL > ASSIGN STDERR, <event message collector>
TACL > RUN SERVER /NOWAIT/ [-P <port number>]
```

NOTE Your <event message collector> may be the standard system log, $0, or a virtual process, such as $VHS.

Example In this example, the Collector listens on port 12345.

```
TACL > ASSIGN STDERR, $0
TACL > RUN SERVER /NOWAIT, NAME $COLL/ -P 12345
```

When you start the Collector, it references a default TCP/IP process($ZTC0). You can change this to the process of your choice by running a DEFINE statement before you start your collector.

```
TACL > ASSIGN STDERR, $0
TACL> DEFINE =TCPIP^PROCESS^NAME, FILE $ZTC8
TACL > RUN SERVER /NOWAIT, NAME $COLL/ -P 12345
```

See the *Reference Guide* for details about the Collector parameters.

Monitoring Collector

Collector event messages are output to the ggserr.log file. You can view this file using the GGSCI VIEW GGSEVT command.

Security considerations

The user ID under which the Collector is started determines whether or not target files can be written and purged. Ensure that the ID has the proper system access to the files and directories written by the Collector.

Collecting between Open Systems and NonStop

Event messages created by the Collector and Replicat on Windows and UNIX systems can be extracted and sent back to EMS on NonStop systems. This feature enables centralized viewing of GoldenGate messages across platforms.

To collect events from other systems:

1. Run Collector on NonStop to collect and distribute EMS messages. For each EMSCLNT process, run one Collector process.

   The following example runs Collector and outputs its messages to $0.

   ```
   TACL> ASSIGN STDERR, $0
   TACL> RUN SERVER /NOWAIT/ –p 7880
   ```

2. Run the EMSCLNT utility on the remote target. EMSCLNT reads a designated error log and runs indefinitely, waiting for more messages to send. When EMSCLNT receives a message, it sends the message to a collector process on NonStop.

   See the examples for running EMSCLNT on open systems for syntax information.

Running EMSCLNT on Open Systems

Example This UNIX example reads the file ggslog.err for error messages. Error messages are sent to the collector to the NonStop at IP address 13.232.123.89 listening on port 7850. The Collector on NonStop writes formatted messages to EMS Collector $0.

```
> $emsclnt –h 13.232.123.89 –p 7850 –f ggserr.log –c $0
```

Example The Windows example below (from the DOS prompt) reads the file d:\ggserrs\log.txt for error messages. Error messages are sent to the collector on host ggs2 listening on port 9876. The Collector on NonStop writes formatted messages to EMS Collector $P0.

```
> emsclnt –h ggs2 –p 9876 –f d:\ggserrs\log.txt –c $P0
```

Argument	Description
`–h ggs2`	The node on which the collector is being run. Can be a name or IP address. This is a required parameter.
`–p 9876`	The port where the collector is listening for messages. This is a required parameter.
`–f d:\ggserrs\log.txt`	The error file where EMSCLNT retrieves error messages. This is a required parameter.
`–c $P0`	The collector where EMS messages should be written on the NonStop (default is $0).

CHAPTER 7
Configuring Change Synchronization

Online change synchronization extracts data and transmits it to a target. This chapter shows you how to prepare Extract, Replicat, and Logger, as well as work with parameter files, which control Extract, Replicat and Logger behavior.

Introduction

If your application is TMF-protected, perform change synchronization using Extract and Replicat. Non-TMF applications use Logger and Replicat to perform the same functions. You can configure GoldenGate to process changes from the following sources:

- A TMF audit trail.
- An intermediate GoldenGate trail, created either by Logger or Extract.
- Directly from entry-sequenced files, or from BASE24, TLF or PTLF files.
- GoldenGate logs generated from non-TMF applications.

Change synchronization for TMF applications

TMF application change synchronization requires at least one Extract group, one trail, and one Replicat group.

Configuring Extract

To configure and run Extract, you must create an Extract group and an Extract parameter file.

1. Start GGSCI:

```
TACL> RUN GGSCI
```

2. Add an Extract group. Specify CPU and priority:

```
GGSCI> ADD EXTRACT EXTORD, BEGIN NOW, CPU 1, PRI 160
```

3. Create the Extract parameter file:

```
GGSCI> TEDIT PARAMS EXTORD
```

Normally, the name of the parameter file is the same as the process group name. For more information on parameter guidelines, see “Working with parameter files” on page 94.

Figure 21 Sample Extract parameter file

```
-- Extract parameter file for
-- TCUSTMER and TCUSTORD changes
EXTRACT EXTORD
GETROLLBACKS
EXTTRAIL \LA.$DATA03.JDSDAT.ET
TABLE $DATA11.JDSSOU.TCUSTMER;
TABLE $DATA11.JDSSOU.TCUSTORD;
```

Configuring trails

Based on considerations such as performance, hard disk constraints, and data throughput speed, you can specify where you want your trails to reside. For example, if you are concerned about disk space in your Extract environment, you may choose to create your trails on the system where Replicat is installed.

To configure your trail, you must create an empty trail using GGSCI, then start its associated process. You can test the trail by checking to see data is being written to it. If an INFO ALL command shows data being written to your trail, it is configured correctly.

To add a GoldenGate trail:

1. Determine if your trail will run locally or remotely. Base this decision on performance considerations vs. data throughput speed.
2. If you are not at the GGSCI prompt, start GGSCI.

   ```
   TACL> RUN GGSCI
   ```

3. Add your trail using the following commands:

   ```
   GGSCI> ADD EXTTRAIL <trail path>, EXTRACT <extract group>, [<trail size
   limit>], [<limit of files waiting to be written>]
   GGSCI> ADD RMTTRAIL <trail path>, EXTRACT <extract group>, [<trail size
   limit>]
   ```

   For example, a local trail would read:

   ```
   GGSCI> ADD EXTTRAIL \LA.$DATA03.JDSDAT.ET, EXTRACT EXTORD, MEGABYTES 5,
   MAXFILES 10
   ```

To test a trail:

1. Issue the GGSCI command INFO ALL.
2. Check to make sure Extract and Replicat are both running, and checkpoint sizes show relative byte addresses.

Configuring Replicat

Replicat gathers data from your trail and delivers it to your target. A Replicat group contains the named Replicat itself, a Replicat parameter file, and checkpoint groups, as desired.

To configure Replicat:

1. From GGSCI, create a Replicat group:

   ```
   GGSCI> ADD REPLICAT <group name>, EXTTRAIL <trail name>
   ```

   For example:

   ```
   GGSCI> ADD REPLICAT REPORD, EXTTRAIL $DATA03.JDSDAT.ET
   ```

2. Launch a NonStop text editor to create a Replicat parameter file:

   ```
   TACL> TEDIT PARAMS REPORD
   ```

3. Enter your parameters as desired.

Normally, the name of the parameter file is the same as the process group name. For more information on parameter guidelines, see "Working with parameter files" on page 94.

Figure 22 Sample Replicat parameter file

```
-- Replicat parameter file for replicating
-- TCUSTMER and TCUSTORD changes

REPLICAT REPORD
HANDLECOLLISIONS
PURGEOLDEXTRACTS
ASSUMETARGETDEFS
DISCARDFILE \LA.$DATA11.GGSDISC.REPORD, PURGE
MAP \LA.$DATA11.GGSSOU.TCUSTORD,
TARGET \NY.$DATA03.GGSTAR.TCUSTORD;
MAP \LA.$DATA11.GGSSOU.TCUSTMER,TARGET \NY.$DATA03.GGSTAR.TCUSTMER;
```

4. Start GGSCI, then start the Replicat you just configured.

   ```
   GGSCI> START REPLICAT REPORD
   ```

5. Verify that Replicat is working and receiving data from Extract.

Change synchronization for non-TMF applications

The Logger program, with the GGSLIB run-time library, extracts changed records from files that are not protected by TMF. A Logger process records database updates in a log trail, which feeds data to Replicat. Each Logger process in a set is named $GGLnn, where nn is a sequenced identifier beginning with 00. For example, if you configure two Logger processes, they are named $GGL00 and $GGL01. Each Logger group has one or more Logger process, a parameter file, one or more log trails, and file extraction lists.

Log trails are sets of files, written to disk, that hold data extracted and sent to a particular Logger. Each log trail is owned by one Logger process. The parameter file holds specific volume locations and the number and size of each log file.

A log trail's name is comprised of the volume and subvolume where GoldenGate Logger is installed, the process prefix, and a series of letters and numbers that grow depending on the number of log trails. For example, $DATA1.GLOGGGL.AA000000 means GoldenGate Logger is installed on volume Data1, subvolume GLOG, the process prefix is GGL, and the trail itself is called AA000000.

To configure and run change synchronization for non-TMF applications, you must:

1. Create the LOGPARM parameter file.
2. Configure Logger and GGSLIB with the ADD LOGGER command.
3. Start Logger.
4. Bind GGSLIB to the non-TMF application.

Creating the LOGPARM file

Just as Extract and Replicat are controlled by parameter files, so is Logger. Unlike either Extract or Replicat, you must create the LOGPARM before you add your Logger to GoldenGate Manager.

To create a Logger parameter file:

1. Start GGSCI.

   ```
   TACL> VOLUME <Enter the volume where GoldenGate is installed>
   TACL> RUN GGSCI
   ```

2. Enter Logger and GGSLIB parameters into the LOGPARM file.

   ```
   GGSCI> TEDIT PARAMS LOGPARM
   ```

3. Add required parameters, including:

   ❍ LOG entries. Each LOG entry describes both a process and a log trail in which the process records database updates. Each log process writes to exactly one log trail.

   ❍ FILE parameters, which specify one or more files to be extracted by the current log process (the current log is the first log entry preceding the FILE entry). The FILE entry may be a wildcard. In addition, FILE may specify compression of update records.

   ❍ EXCLUDEFILE parameters, which specifically exclude a file from a list for a particular log if it has been included with FILE.

   ❍ The primary and backup CPUs in which the particular log process will run.

   ❍ The priority at which the Logger process will run (PRIORITY).

Logger parameters are detailed in the *Reference Guide*. The following section outlines a sample Logger parameter file.

Sample LOGPARM file

This sample parameter file configures two log processes $GGL00 and $GGL01. These process names have been explicitly set with the PROCESS parameter, but when not set the names default to $GGLnn. The system will increment nn from 00 so the default will generate the same process names in this instance.

NOTE Parameter position is important. As soon as a log entry is specified with the log parameter, it becomes the current log. Parameters entered below the current log parameter apply only to the current log. For instance, in the following example, all parameters after the LOG $D3.GGSLOG.AA and before the LOG $D15.GGSLOG.BB entry apply to LOG $D3.GGSLOG.AA.

Figure 23 Sample Logger parameter file

```
-- Logger configuration for two Loggers

LOG $D3.GGSLOG.AA, PROCESS $GGL00, MEGABYTES 500, NUMFILES 10, SECURE
"NUUU"
CPU 9,4
FILE $DATA4.*.*
FILE $DATA5.*.*
EXCLUDEFILE $DATA4.REPORTS.*
EXCLUDEFILE $DATA4.DAT.TRANSFL

LOG $D15.GGSLOG.BB, PROCESS $GGL01, MEGABYTES 100, NUMFILES 10, SECURE
"NUUU"
CPU 3,2

FILE $*.*.*

EXCLUDEFILE $DATA4.REPORTS.*
```

Logger GGL00:

- Creates a log trail $D3.GGSLOG.AA that contains 10 files each sized at 500 megabytes (for a total of 5,000 megabytes). The file names will be AA000000, AA000001, through AA000009. As new files are required, the oldest one is recycled and takes the next sequence number; in this case, AA000000 will become AA000010. File space is pre-allocated by the GGSCI and Manager processes.
- Configures $GGL00 to run on CPU 9, with backup CPU 4.
- Specifies that data written by the application in $DATA4 will be logged to the log trail $D3.GGSLOG.AA.
- Specifies that data written by the application in $DATA5 will be logged to the log trail $D3.GGSLOG.AA.
- Excludes $DATA4.REPORTS.* from being logged to AA.
- Excludes $DATA4.DAT.TRANSFL from being logged to AA.

Logger GGL01:

- Creates a log trail $D15.GGSLOG.BB that contains 100 files each sized at 10 megabytes (for a total of 1,000 megabytes). The file names will be BB000000, BB000001, through BB000009. These files are recycled when needed.
- Configures $GGL01 to run in CPU 3, with backup CPU 2.
- Specifies that data written by the application to files in any location should be written to the BB log trail, with the exception of $DATA4.REPORTS.* and any data already captured by $GGL00 (in this case, $DATA4.*.* and $DATA5.*.*). $DATA4.DAT.TRANSFL will be captured in BB since it was implicitly included in $*.*.* and excluded nowhere for this logger.

Configuring Logger and GGSLIB

Execute the ADD LOGGER command to process the configuration in LOGPARM. This step establishes a configuration for both Logger and GGSLIB and pre-allocates disk files for each Logger process to use for logging database updates.

Before starting Logger, GoldenGate must process and store its configuration. This step pre-allocates file space for each log trail to ensure extracted records can be stored.

To process the Logger configuration, enter the following command.

```
GGSCI> ADD LOGGER
```

Starting Logger

To start Logger:

1. Start GGSCI.
2. Enter START LOGGER.

   ```
   GGSCI> START LOGGER
   ```

   By default, this command starts the logger group $GGLnn. If, for example, you have three LOG entries in the LOGPARM file, START LOGGER starts three processes, named $GGL00, $GGL01 and $GGL02.

Using macros to bind GGSLIB to a non-TMF application

Use the GGSCI BIND PROGRAMS command to link the GGSLIB library to your non-TMF application. This step also binds GGSLIB to any existing user libraries the application may already invoke.

```
TACL > RUN GGSCI
GGSCI> BIND PROGRAMS
```

BIND PROGRAMS prompts for a list of programs to bind with GGSLIB. GGSCI analyzes this list to see which files are already bound and which ones it needs to bind.

In this context, *bound* means that GGSLIB runs as a user library in the application program (BIND CHANGE LIBRARY GGSLIB in <program> is executed). GGSLIB is not literally bound with the application program. If a program already invokes a user library, that library is literally bound with GGSLIB to create a new library. The library will have the same name as the old user library.

Building GGSLIB

GGSLIB, built as part of installation, contains the BASELIB module that intercepts Guardian function calls made by the application. GGSLIB also contains C, CRE and COBOL runtime libraries that call Guardian functions. When bound to GGSLIB, these libraries attempt to call the operating system function, but actually call the GoldenGate function instead. GGSLIB in turn calls the intended operating system function transparently to the application. GGSLIB uses a shared extended memory segment for efficient configuration storage, and maintains a private memory segment for working storage variables.

Without the presence of these libraries, the C and COBOL run-time libraries would be called at the operating system level and would bypass GoldenGate intercept functions.

Therefore, build these libraries carefully. Keep the following libraries up-to-date with your latest operating system release and related IPMs. Not all of these libraries are required in the GGSLIB build if your application does not run COBOL74, COBOL85 or C routines. It is recommended, however, to bind each of these components that exist on your system into GGSLIB.

Library	Function
`$SYSTEM.ZCOBOLRT.CLIBOBJ`	COBOL74 routines
`$SYSTEM.ZCOB85RT.C8LIB`	COBOL85 routines
`$SYSTEM.ZCRERTL.CFELIB`	Common Run-time Environment
`$SYSTEM.SYSTEM.CRELIB`	Common Run-time Environment
`$SYSTEM.SYSTEM.COBOLLIB`	More COBOL85 routines

To build a new version of GGSLIB, issue the following command from TACL.

```
> RUN BUILDMAC
```

In some Guardian releases there are insignificant conflicts between functions that appear in more than one of the above libraries. You can safely ignore the resultant BIND warnings during the build.

Private memory and stack space

GGSLIB routines minimize stack space requirements. By doing so, programs are ensured that normal activities will have enough stack room left for themselves.

For its own working space, GGSLIB allocates a small private memory segment to handle in-transit I/O buffers and keep its own state variables.

Alternate methods of binding GGSLIB to an application

There are alternatives to using the GoldenGate macros (e.g. NLDLIB) to bind the GoldenGate intercept library to your application. These alternatives may vary depending on your NonStop environment.

For non-native mode systems, a type 100 object file is produced using the TAL, COBOL, or C compilers. Native-mode S systems use PTAL, NMCOBOL, NMC compilers to create type 700 objects. Native mode Itanium systems use EPTAL, ECOBOL or CCOMP to compile type 800 objects.

Using the ?SEARCH directive

One way to connect the application and the GoldenGate intercept library is to use the ?SEARCH directive in the compile. This copies the library into the application object file. The drawback to this method is that an upgrade to the GoldenGate application or the operating system will not be picked up by the built-in modules of these programs. A recompile is required to replace the modules.

Non-native environments

The intercept library can be bound to application programs in non-native environments by using:

- NonStop's BIND utility

  ```
  BIND CHANGE LIBRARY <$vol.svol.library> IN <application object>
  ```

- A /LIB / parameter in the run statement

  ```
  RUN <application object>/LIB <$vol.svol.library>/
  ```

- SET SERVER GUARDIAN-LIB parameter if it is a Pathway server

Native mode S systems

For the native mode S environment, application programs should not be compiled RUNNABLE. The programs build-in the operating system shared runtime modules during the RUNNABLE compile, leaving GoldenGate no opportunity to intercept I/O calls.

To enable the GoldenGate intercept library in this environment, operating system modules ZCOBREL and ZCREREL must be checked out from the Distributed Systems Management/Software Configuration Manager (DSM/SCM) and bound to the application. When the application program is compiled, it is run through a Native Link Editor (NLD) step to set the called addresses to functions within the intercept library instead of the operating system.

The steps are:

- Run the compile without the RUNNABLE option
- Execute the following command entering <temp object> as the output from the compiler, <application object> as the runnable application program, and <$vol.svol.library> as the GGSLIB library object file.

  ```
  NLD <temp object> -o <application object> -libname <$vol.svol.library>
  ```

NOTE The install macro must be run before the NLD step. It prompts for the locations of the *REL files and correctly builds the library.

Native mode Itanium systems

The native mode Itanium system does not require any special steps. The intercept library can be bound to the application by any of the following.

- Using the TNS/E Link edit (ELD) utility change command

  ```
  ELD -CHANGE LIBNAME '<$vol.svol.library>' <application object>
  ```

- A /LIB / parameter in the run statement

  ```
  RUN <application program>/LIB <GoldenGate library>/
  ```

- Using the server parameter GUARDIAN-LIB.

Libraries for native applications

If your NonStop environment is running in native mode, you may decide to invoke native mode libraries so processes run more efficiently. You must use native mode modules and libraries if you have invoked the encryption or compression capabilities of GoldenGate. GoldenGate provides a TACL macro, NLDLIB, for building the following native libraries:

- **BASELIBR**: A relinkable, native version of BASELIB, a module that intercepts function calls made by the application.

- **GGSSRL**: A native version of BASELIB for use as a user shared runtime library (SRL) on G06 operating systems.
- **GGSDLL**: A native version of BASELIB for use as a dynamically linked library (DLL) on H06/J06 operating systems.
- **GGSLIBR**: A relinkable, native BASELIB containing CRE and COBOL SRLS.

NOTE Applications running on H06/J06 that include native C, native COBOL, and pTAL require two intercept libraries. The one to be linked to the C and COBOL applications should reference the COBOL and CRE dynamic link libraries, and the one for pTAL should not. This is due to the limitation that pTAL does not perform the necessary initialization of the run-time environment.

Running NLDLIB

Running the NLDLIB macro allows you to create these libraries and combine them with the native mode GoldenGate BASELIBN library and certain Guardian system libraries. You can run NLDLIB as part of your initial installation routine or on its own.

1. Run the following:

```
TACL> RUN NLDLIB
```

2. The NLDLIB macro runs, and you are asked if you want to include a user library. Reply either **Y** or **N**.

```
*** Running NLDLIB to build GGSLIBR & GGSSRL ***
----------------------------------------------------------------
NLDLIB builds the native relinkable libraries GGSLIBR and GGSSRL
       A Native mode GGSLIB needs the re-linkable versions of some
       Tandem SRLs. ZCREREL and ZCOBREL. If Native cobol support is not
       required then ZCOBREL may be omitted.
If Native Cobol support is required then you have to compile the
       Cobol program and search in GGSLIBR
A Native mode GGSSRL will be built for a User Library for Dynamic
linking

Enter X at any prompt to EXIT
----------------------------------------------------------------

Do you want to include your own User Library (Y/N) : N
```

3. You are asked if you want to change the location of the AUDCFG.

```
Do you want to change the location for the AUDCFG segment(Y/N) : N
```

   If you respond with yes, it prompts you for the new default location ($VOL.SUBVOL) of the AUDCFG segment.

4. You are asked if you want to include ZCREREL. It is recommended to enter **Y** to include it unless you know your applications do not use these libraries.

```
Do you want to include ZCREREL (Y/N) : Y
```

NLDLIB searches for ZCREREL locations and displays them.

```
Looking for ZCREREL ..
$SYSTEM.SYS02.ZCREREL $SYSTEM.SYS03.ZCREREL $SYSTEM.SYS04.ZCREREL
$DATA09.R1269G09.ZCREREL $DATA09.R8431G09.ZCREREL
```

5. You are asked to enter the location of the ZCREREL to use.

   ```
   Enter location of ZCREREL : $SYSTEM.SYS03.ZCREREL
   ```

6. You are asked if you want to include ZCOBREL. It is recommended to enter **Y** to include it unless you know your applications do not use these libraries.

   ```
   Do you want to include ZCOBREL (Y/N) : Y
   ```

   NLDLIB searches for ZCOBREL locations and displays them.

   ```
   Looking for ZCOBREL ..
   $SYSTEM.SYS02.ZCOBREL $SYSTEM.SYS03.ZCOBREL $SYSTEM.SYS04.ZCOBREL
   $DATA09.R8107G09.ZCOBREL $DATA09.R8108D46.ZCOBREL
   ```

7. You are asked to enter the location of the ZCOBREL to use. If you plan to use COBOL, enter a location; otherwise, you can press **Enter** to bypass this prompt.

   ```
   Enter location of ZCOBREL : $SYSTEM.SYS03.ZCOBREL
   ```

   Note: If you omit a ZCOBREL location, you are prompted to confirm your choice:

   ```
   Omit ZCOBREL (y/n):Y
   ```

   NLDLIB builds the GGSLIBR and GGSSRL libraries, displaying a series of informational messages and the names of the files that were created.

8. When the libraries are built, add the new relinkable library to your program object using the LINK PROGRAMS command in GGSCI. You could also run the program as follows:

   ```
   TACL> RUN <your program name> /LIB <new library name>/
   ```

You can also run the NLDLIB macro from the TACL prompt providing arguments in the command line. This is not recommended, however, as it may produce unexpected results. Interactive responses help ensure the appropriate options for your environment.

Removing a Library

To remove the GoldenGate library from your application, run your program with an empty LIB parameter.

Example

```
TACL> RUN <your program name> /LIB/
```

Activating authorization of bound libraries

GoldenGate's SFGEXIT module can be added to Safeguard to produce a warning for any program that opens non-audited files for update and does not have the GoldenGate intercept library bound to it. See "Authentication for bound programs" on page 48.

NOTE Opens on SQL tables, unstructured files, and TMF protected files are always ignored. Opens from processes on remote nodes are also ignored.

When the program is added, the authorization event can be enabled and optional PARAM-TEXT arguments can be supplied.

The syntax for the ADD within Safecom is:

```
=ADD EVENT-EXIT-PROCESS OPENCHECK PROG $<vol>.<subvol>.SFGEXIT
   [, PNAME <process name>]
   [, ENABLE-AUTHORIZATION-EVENT {ON | OFF}]
      [, ENABLE {ON | OFF}]
   [, PARAM-TEXT
      [, DETAIL] |
      [, OSOPENSUMMARY| OSOPENDETAIL | NOOSOPENS]
      [, AUDCFG <file name> [REJECT]]
      ]
```

Option	Description
`ENABLE-AUTHORIZATION-EVENT ON | OFF`	The authorization event can be set to ON during the ADD of the event. If this is not included, ENABLE-AUTHORIZATION-EVENT will default to OFF and can be set to ON using an ALTER command.
`ENABLE ON | OFF`	If ENABLE-AUTHORIZATION-EVENT is set to ON during the ADD, ENABLE can also be set to ON. If it is not set, ENABLE will default to OFF and can be set to ON using the ALTER command.
`PNAME <process name>`	Optionally a logical process name can be entered.
`PARAM-TEXT`	PARAM-TEXT has the following options: ◆ `DETAIL` DETAIL specifies that a message should be logged to EMS every time a user application that is not bound to the GoldenGate library opens a file for update. The default is to display a message only the first time the application opens a file for update. **Note:** Use the DETAIL option with care. It may produce a large number of EMS messages due to OPENS for alternate key files and partitions. ◆ `OSOPENDETAIL | OSOPENSUMMARY | NOOSOPENS` OSOPENDETAIL and OSOPENSUMMARY both specify that messages will be logged for OS processes (programs started from $SYSTEM.SYSTEM and $SYSTEM.SYSnn). NOOSOPENS will not log warnings for this type of process. The default is NOOSOPENS. OSOPENDETAIL further specifies that messages should be logged to EMS every time a process that is not bound to the GoldenGate library opens a file for update. OSOPENSUMMARY displays a message only the first time the process opens a file for update.

Option	Description
	◆ AUDCFG <file name> [REJECT] Identifies the Logger audit configuration file as <file name>. When this option is used, only files matching an entry in the indicated AUDCFG file are acted upon. If a file not in the AUDCFG is opened for update, SFGEXIT replies NO RECORD without applying any processing. If REJECT is specified, an open will be refused when a program does not have GGSLIB bound in and it tries to open a file listed in AUDCFG. **Note:** Using wildcards in the file list may generate unexpected number 48 errors if it causes the tracking of files that the application would not normally open.

Managing the authorization event

Adding and verifying the authorization event

The following steps show examples that add, set options, check the status, and remove the authorization event.

1. To add the authorization program, access SAFECOM and enter the ADD statement as shown in the example below.

```
>SAFECOM
=ADD EVENT-EXIT-PROCESS OPENCHECK PROG $DATA1.GGS.SFGEXIT, PNAME $ZSEEP,
ENABLE-AUTHORIZATION-EVENT ON, ENABLE ON, PARAM-TEXT DETAIL
```

2. The INFO command can be used to verify the addition, check the location of the program, check the status of the event, and review the PARAM-TEXT options:

```
=INFO EVENT-EXIT-PROCESS OPENCHECK

EVENT-EXIT-PROCESS  OPENCHECK
   ENABLED = ON
   RESPONSE-TIMEOUT =    5 SECONDS
   ENABLE-AUTHORIZATION-EVENT  = ON
   ENABLE-PASSWORD-EVENT       = OFF
   PROG  = $DATA1.GGS.SFGEXIT
   LIB   = * NONE *
   PNAME = $ZSEEP
   SWAP  = * NONE *
   CPU   =       2
   PRI   =     155

PARAM-TEXT = DETAIL
```

3. To remove the OPENCHECK event, you must first turn off the activation with the ENABLE OFF.

```
=ALTER EVENT-EXIT-PROCESS OPENCHECK, ENABLE OFF
=DELETE EVENT-EXIT-PROCESS OPENCHECK
```

4. To exit SAFECOM:

   ```
   =EXIT
   ```

Using different PARAM-TEXT options

Other examples of setting options when adding the authorization event are shown below.

1. The following adds an OPENCHECK event that will issue a warning each time the application opens a file that does not have the intercept library bound. It also issue warnings for each openfor programs that are run from $SYSTEM.SYSTEM and $SYSTEM.SYSnn.

   ```
   =ADD EVENT-EXIT-PROCESS OPENCHECK PROG $DATA1.GGS.SFGEXIT, PARAM-TEXT
   DETAIL, OSOPENSDETAIL
   ```

2. The following adds an OPENCHECK event that will issue only one warning for each file without an intercept library, evaluate only files listed in the audit configuration file $DATA1.GGS.AUDCFG, and not include programs that are run from $SYSTEM.SYSTEM and $SYSTEM.SYSnn.

   ```
   =ADD EVENT-EXIT-PROCESS OPENCHECK PROG $DATA1.GGS.SFGEXIT, PARAM-TEXT
   AUDCFG $DATA1.GGS.AUDCFG
   ```

Getting the current status of the authorization event

Additional examples of monitoring the process are shown below.

1. The following SEND <process> GETSTATS command retrieves statistics from a running authorization event.

   ```
   GGSCI (\NY) 2445> SEND $ZSEEP, GETSTATS

   \NY.$ZSEEP Stats at 2007-10-15 15:14:01.770337
   Started 2007-10-15 15:07:47.444913 CPUTime 0:00:00.007707  (PerOp 31)
   Audcfg \NY.$data01.zlogdat.audcfg  Modtime 2007-09-25 12:10:00.845007
   PoolGets            7    PoolPuts         0
   GGSRequests        10    Other            0
   SFGRequests       205    Total          245
     Access          205    RemoteNode       0
     NonDisk          50    SQL              0
     Open            154    Readonly       129
     Audcfg Check      2    Found            2    Excluded             0
     Diskfiles       154    Unstruct         0
     TMF Audited       0    SQL Tables       0
     ProcessInfo       0    Cached           0    Errors               0
     FileInfo          2    Cached           0    Errors               2
     GGSProgs          0    SystemProgs      0
     Reported          0    OpensDenied      0
   Hash Stats
     Buckets        7919
     Entries           2    Lookups          2
     Collisions        2    Depth            0
   ```

2. The following SEND <process> PROCESSINFO command retrieves information on the process.

```
GGSCI (\NY) 19886> send $zseep, processinfo 3,1192

3,1192 \NY.$QA01.BV95014.REPLICAT GGS Code Has Lib
```

3. The following example errors were retrieved by the SEND <process> GETERRORLIST command.

```
2007-10-15 15:12:49.911382
 FILE_GETINFOLISTBYNAME_ error 11 on \NY.$SYSTEM.SYS07.INSPLOG
2007-10-15 15:13:52.254180
 FILE_GETINFOLISTBYNAME_ error 11 on \NY.$DATA01.QA.TESTFILE
```

Working with parameter files

Parameters give you complete control over all aspects of GoldenGate, such as:

- Data selection, mapping, and transformation.
- Replication.
- Error resolution.
- Logging
- Status and error reporting.
- System resource usage.
- Startup and runtime behavior.

There can be only one active parameter file for each Manager, Extract, or Replicat. There are two types of parameters: global and file-specific.

- Global parameters apply to all tables specified in the parameter file for synchronization. Some global parameters affect process behavior while others affect such things as memory utilization and so forth.
- File or table-specific parameters control processing for tables specified with a FILE, TABLE or MAP statement. Table-specific parameters enable you to designate one set of processing rules for some tables, while designating other rules for other tables. There are two implementations for file-specific parameters:
  - Toggling the parameter on and off around one or more FILE, TABLE or MAP statements.
  - Adding the parameter within MAP statement so it applies only to that table or file.

Some parameters, such as HANDLECOLLISIONS/NOHANDLECOLLISIONS can be included in a MAP statement or toggled ON and OFF. Others can be implemented using only one of the methods. For further details, see the *Reference Guide*.

The ordering of parameters in a parameter file can be important.

- A global parameter can appear anywhere in the parameter file, and it should only be listed in the file once. When listed more than once, only the *last* instance of the parameter is active. All other instances are ignored.
- Table-specific parameters take effect in the order that each parameter is listed in the file.

Table 4 Basic Extract and Replicat parameter files

Sample Extract parameter file	Sample Replicat parameter file

`EXTRACT NYTOLA`
`DISCARDFILE =DISCARD_FILE, PURGE`
`EXTTRAIL $DATA1.EXTDAT.XX`
`FILE $DATA2.FINANCE.ACCOUNTS;`
	`REPLICAT NYTOLA`
	`DISCARDFILE =$DATA.GGSDISC.NYTOLA, PURGE`
	`ASSUMETARGETDEFS`
	`MAP $DATA2.FINANCE.ACCOUNT,`
	`TARGET $BACK.FINANCE.ACCOUNTS;`

Creating a parameter file

From the directory where GoldenGate is installed, create a parameter file using the NonStop text editor. Normally, the name of the parameter file is the same as the process group name. For example, if you created the Extract group ADD EXTRACT NYTOLA, you would create your parameter file by entering TEDIT PARAMS NYTOLA.

To create a parameter file through GGSCI

1. From the directory where GoldenGate is installed, run the GGSCI command-line user interface.
2. In GGSCI, issue the following command to open the default text editor.

   ```
   GGSCI> EDIT PARAMS <group name>
   ```

   Where: <group name> is either MGRPARM (for the Manager process), LOGPARM or the name of the Extract or Replicat group for which the file is being created. The name of an Extract or Replicat parameter file must match that of its process group.

   Examples:

   ❍ The following creates or edits the parameter file for an Extract group named EXTORA.

   ```
   GGSCI> EDIT PARAMS EXTORA
   ```

   ❍ The following creates or edits the parameter file for the Manager process.

   ```
   GGSCI> EDIT PARAMS MGRPARM
   ```

   ❍ The following creates or edits the parameter file for the Manager process.

   ```
   GGSCI> EDIT PARAMS LOGPARM
   ```

3. Using the editing functions of the editor, enter as many comment lines as you want to describe this file, making certain that each line is commented out by two hyphens (--). As an alternative, you can use the COMMENT parameter, which causes everything on the same line as the COMMENT parameter to be ignored. The syntax for COMMENT is:

   ```
   COMMENT <comment text>
   ```

   NOTE Do not put a dash or pound symbol before the COMMENT keyword. Do not use COMMENT if any column names in the tables contain the word "comment." Instead, use double hyphens (--).

4. On non-commented lines, enter the parameters for your synchronization configuration, starting a new line for each parameter statement.

For parameters that accept table names, you can use an asterisk (*) wildcard to match any number of characters.

Parameters have the following general syntax:

```
<parameter> <argument> [, <option>] [&]
```

Where:

- <parameter> is the parameter name.
- <argument> is a required argument for the parameter. Some parameters take arguments, while others do not. Separate all arguments with commas, as in the following example:

  ```
  USERID ggs, PASSWORD ggs123
  RMTHOST sysb, MGRPORT 8040
  RMTFILE $DATA5.GGS9000.GGSDAT.C1, PURGE
  ```

- <option> is an optional argument.
- & enables you to continue a parameter's arguments on another line. Place it at the end of the line to be continued.

NOTE Parameters that terminate with a semicolon, such as MAP, do not require ampersands (&) to span more than one line.

5. Save and close the file.

Storing parameter files

By default, parameter files are stored in the GGSPARM subvolume. If you are not going to use the default location, create the new location before starting GoldenGate. You can change this default location using the ADD DEFINE parameter in the GLOBALS parameter file, which would look like the following example.

```
TACL> ADD DEFINE =GGS_PARAMS, CLASS DEFAULTS, VOLUME $VOL.SUBVOL
```

Once paired with a process, a parameter file must remain in its original location for GoldenGate to operate properly.

Viewing a parameter file

You can view a parameter file by issuing the GGSCI VIEW PARAMS command.

Syntax `VIEW PARAMS <file name>`

VIEW PARAMS displays the file. You can scroll through its contents in a variety of ways, such as:

Command	Description
`<return>, n`	Next page
`/<string>`	Search for next occurrence of <string> in file
`<number>`	Go to line indicated by <number>

Command	Description
`l`	Go to last page of file
`b`	Go backwards one page in file
`q`	Quit display
`h`	Help

Your *Reference Guide* has a complete list of commands.

Changing a parameter file

The procedure for changing a parameter file depends on its dependent process. Manager parameters can be changed on the fly; Extract and Replicat files cannot be changed until Extract and/or Replicat is stopped.

To apply some Manger changes on the fly, issue the REFRESH MANAGER command. The changes are implemented automatically. However, some startup parameters, such as TCP/IP PROCESS NAME cannot be refreshed in this way. Changes to these types of parameters require that you stop Manager, make your changes, then restart the process.

To change an Extract or Replicat parameter file, stop the associated process, then edit the file with the NonStop text editor. Make your changes, then verify them with the CHECKPARAMS parameter as described in "Verifying a parameter file" on page 97.

Using OBEY and Macros in parameters

You can leverage existing parameter files through the GoldenGate macros and the OBEY command. To simplify the process, you can use GoldenGate macros for a variety of operations, including implementing multiple uses of a statement, consolidating multiple commands, or invoking other macros. You also can use OBEY to direct GoldenGate to retrieve parameter settings from another parameter file. Upon encountering OBEY, GoldenGate processes the parameters from the other file and then returns to the current file to process any remaining instructions.

See "Configuring Custom Operations" on page 99 for more information about using macros and OBEY files.

Verifying a parameter file

Use the following procedure to confirm that the syntax in an Extract or Replicat parameter file is correct:

1. Include CHECKPARAMS in the parameter file.
2. Start the associated process.
3. GoldenGate audits the syntax and writes the results to the report file or screen. View the report by issuing the following:

   ```
   GGSCI> INFO <process type> <group name>
   ```

For example:

```
GGSCI> INFO REPLICAT REPCUST
```

- If the syntax is correct, remove the CHECKPARAMS parameter and start the process again to begin processing.
- If the syntax is wrong, edit the file to correct the syntax based on the report's findings, and then start the process again.

Substituting a parameter

It is possible to assign different values to a parameter within a parameter file. One-off change synchronization runs that require specific parameters can execute with the same parameter file as your default change synchronization routine; any difference in parameter requirements is handled by parameter substitution. This minimizes your need for multiple parameter files.

To include a run-time parameter within the parameter file, precede any intended parameter name with a question mark. Then, prior to running the Extract process, use the TACL PARAMS command to pass the value.

Example Parameter file contents:

```
SOURCEISFILE
EXTFILE ?extfile
TABLE ?tabname, WHERE (REGION = "?region");
```

When you are ready to run your special data run, specify the following from your TACL prompt:

```
TACL> PARAM EXTFILE $DATA2.GGS.EXTFILE
TACL> PARAM TABNAME $DATA3.MYDB.ACCOUNTS
TACL> PARAM REGION EAST
TACL> RUN EXTRACT /IN PARMFL/
```

Extract will interpret the parameter as follows:

```
SOURCEISFILE
EXTFILE $DATA2.GGS.EXTFILE
TABLE $DATA3.MYDB.ACCOUNTS, WHERE (REGION = "EAST");
```

NOTE A question mark can also be used as a wildcard so care should be exercised in using PARAMS and wildcards together. The program will process parameter substitutions first, before evaluating wildcards. It cannot distinguish, however, between ?DATA as a parameter and ?DATA as a wildcard, so it is important that the user selects parameter names that are never used as part of an actual file name.

CHAPTER 8
Configuring Custom Operations

Custom operations make it possible for you to tailor GoldenGate to the specific needs of your organization. You can write C or COBOL routines and call them with GoldenGate user exits. You can also save frequently used GoldenGate routines as macros then call the macros from within Extract or Replicat parameter files. You can use OBEY files to access frequently used GoldenGate parameters.

User exits

User exits allow you to extend and customize the functionality of Extract and Replicat. At different points during Extract and Replicat processing, you can invoke COBOL, C or TAL routines to perform an unlimited number of functions. You can also easily add functions to the application and respond to database events almost as soon as they occur without altering production programs. For example, user exits can:

- Perform arithmetic operations, special date conversions or table lookups while mapping from one file format to another.
- Implement record archival functions off-line.
- Respond to unusual database events in custom ways, for example, by sending a formatted e-mail message or paging a supervisor based on some field value.
- Accumulate totals and gather statistics.
- Clean up invalid data.
- Determine the net difference in a record before and after an update.
- Accept or reject records based on complex criteria.
- Normalize a database during conversion.
- Eliminate indexes that exist to identify recently changed records.

Record formats for user exits

User exits expect records to have a specific format. For example, user exits expect:

- Deletes, inserts, and updates to appear in the buffer as full record images.
- Non-compressed data to have no offset or length preceding data.
- Compressed Enscribe and SQL updates to both have the following format:
  `<offset><length><value><offset><length><value>…`

where

- <offset> is the offset into the Enscribe record of the data fragment that changed.

- <length> is the length of the fragment.
- <value> is the data. Fragments can span field boundaries, so full fields are not always retrieved (unless compression is off or FETCHCOMPS is used).
- Enscribe has an I/O type of 11; SQL has an I/O type of 15. All other I/O types for deletes, inserts, and updates are in non-compressed format.

NOTE The above record formats only apply to data sourced from an HP NonStop system.

Creating user exits

To implement user exits:

1. Create a user exit shell routine in C, TAL or COBOL. The user shell routine is the communication point between Extract or Replicat and your routines.
   - **C shell routines.** Shell routines written in C must be named CUSEREXIT and must accept the EXIT-CALL-TYPE, EXIT-CALL-RESULT, EXIT-PARAMS, and EXIT-REC-BUF parameters. These parameters are supplied by GoldenGate in the XLIBC include file.
   - **COBOL shell routines.** Shell routines written in COBOL must specify the ENV COMMON directive, and the PROGRAM-ID of one of the modules must be named COBOLUSEREXIT. The COBOLUSEREXIT program must have a linkage section that contains EXIT-CALL-TYPE, EXIT-CALL-RESULT, EXIT-PARAMS, and EXIT-REC-BUF parameters. These parameters are supplied by GoldenGate in the XLIBCOB copy library.
   - **TAL shell routines.** Shell routines written in TAL must be named TALUSEREXIT and must accept the EXIT-CALL-TYPE, EXIT-CALL-RESULT, EXIT-PARAMS, and EXIT-REC-BUF parameters. These parameters are supplied by GoldenGate in the XLIBTAL include file.

   See the *Reference Guide* for details about both C and COBOL routines.
2. Include Calling Environment Functions to retrieve information such as record buffers and transaction contexts, if necessary. If the user exit is written in C, you must include the USRDECS file. If the exit is written in COBOL, you must furnish a CONSULT directive to either Extract or Replicat. If the exit is written in TAL, you must source the USRDECT file.
3. In any language, create routines to respond to each type of event generated by Extract and Replicat.
4. Compile and bind the shell routine and the routines that respond to individual events, creating the user exit module.
5. Bind the user exit module with Extract or Replicat by running the BINDEXIT macro and creating a custom Extract or Replicat module with a different name. For further information, see “Binding the user exit”.
6. Include the CUSEREXIT, COBOLUSEREXIT or TALUSEREXIT parameter in your Extract or Replicat parameter file.
7. Run the custom Extract or Replicat module.

Binding the user exit

BINDEXIT is an interactive macro that creates a new object file to combine Extract or Replicat with user exit routines. BINDEXIT syntax is similar to:

Syntax

```
TACL> RUN $<volume>.<subvolume>.BINDEXIT [<options>...] [<object type>...]
```

The following information can be entered as options or BINDEXIT will prompt you for it.

Options	Description
`USEROBJ`	The name of the user exit object file.
`NEWOBJ`	The name of the new object file that will contain the exit routines and the Extract or Replicat module. The file must not yet exist, and will be created in the same subvolume as the Extract/Replicat module.
`GGSUBVOL`	The fully defined location of the GoldenGate environment.
`AXCEL | NOAXCEL`	AXCEL runs the NonStop Accelerator program to accelerate the TNS object files. This is the default, but it can be bypassed with NOAXCEL.
`CATALOG`	The SQL Catalog for the SQLCOMP of the program. This information is not required if your database is Enscribe.
`SHOWCMD`	Displays additional information on the BINDEXIT commands during the session.
`HELP`	Display BINDEXIT Help text.

Argument	Description
`<object type>`	The type of file to create, either Extract or Replicat.

BINDEXIT binds your code with the Extract or Replicat code, creating the new object file. BINDEXIT ensures that you included either a CUSEREXIT or COBOLUSEREXIT routine, and that no conflicts exist between your code and the Extract or Replicat module (such as having the same names for different functions). Once the new object file is created, run that file rather than Extract or Replicat.

Figure 24 BINDEXIT Help

```
TACL> RUN BINDEXIT Help
Usage: RUN BINDEXIT [<options>...] [<object type>]
<options> are
USEROBJ NEWOBJ GGSUBVOL AXCEL CATALOG SHOWCMD HELP
<object type> [EXTRACT | REPLICAT]
```

Binding user exits in native mode

If you are running your NonStop environment in native mode, you must bind your native exits using NLDEXIT instead of BINDEXIT. NLDEXIT runs just as BINDEXIT does, and prompts you for the same and some additional information.

Syntax

```
TACL >RUN $<volume>.<subvolume>.NLDEXIT
[<options>...] [<object type>...]
```

Options	Description
`USEROBJ`	The name of the user exit object file.
`NEWOBJ`	The name of the new object file that will contain the exit routines and the Extract or Replicat module. The file must not yet exist, and will be created in the same subvolume as the Extract/Replicat module.
`GGSUBVOL`	The fully defined location of the GoldenGate environment.
`CATALOG`	The SQL Catalog for the SQLCOMP of the program. This information is not required if your database is Enscribe.
`C++VERSION`	If there were any C++ modules used, the version of the C++ compiler; 2 or 3.
`CEXITWITHCOBOL`	Y or N to indicate whether any COBOL modules were used.
`SHOWCMD`	Displays additional information on the NLDEXIT commands during the session.
`HELP`	Display NLDEXIT Help text.

Figure 25 Some of the additional information displayed with NLDEXIT and SHOWCMD

```
-o $DATA1.GGSSRC.TESTREP
$DATA2.TSPAK.XSKLCON $DATA2.TEST.REPR
$DATA2.TSSOBJ.USRESQL
-nostdfiles
-allow_duplicate_procs
-set runnamed on
-set highpin on
-set highrequesters on
-set saveabend on
-set libname $DATA1.GGSSRC.PRIVLIB

$system.system.crtlmain
-obey $system.system.libcobey
NLD - NATIVE MODE LINKER - T6017D45 - 20APR04
(C)1993 Tandem (C)2004 Hewlett-Packard Development Company, L.P.

NLD's command line was:
     \LA.$system.system.nld -stdin
**** INFORMATIONAL MESSAGE **** [20022]:
     The SRL name or archive name specified as 'zcresrl' in a -l, -lib, or
     -import flag was resolved to the SRL named
     '\LA.$SYSTEM.SYS04.zcresrl'.
```

```
...(11 informational messages omitted from this sample)

NLD reported 0 errors.
NLD reported 0 warnings.
NLD reported 12 informational messages.
NLD created the following type of object file:
     \TRILL.$DATA1.GGSSRC.TESTREP (ELF, executable)

NLD Timestamp:  15DEC2005 15:07:30
Elapsed Time:   00:00:06
```

Example The following example creates a new native user exit in Extract.

```
TACL> RUN $DATA2.GGS.NLDEXIT
Creates a new Native Extract or Replicat object file linked with a USEREXIT
module.
Enter X at any prompt to quit.

Enter type of GGS object to create Extract or Replicat:
GGS Object type: <Extract>
Enter $Vol.Subvol for Extract relinkable <location of GoldenGate subvolume>
Enter location of userexit object: <your native compiled C object>
Enter name for new object file: <new native Extract>

Does your C User Exit contain C++ modules (Y/N): <Y>
What version compiler was used for C++ (2/3): <number>
Does your C User Exit contain Cobol modules (Y/N)? <Y>
New Extract file $DATA2.GGS.<Extract name>.<file name> created with user
exits.

SQL Catalog for SQLCOMP (or N to avoid SQL compile): <any SQL catalog subvol>
```

Debugging Replicat user exits

Once you have bound your user exit into Replicat to create a new object, you will want to debug your new code. If you are using TNS Replicat, use the following command:

If your Replicat is in TNS mode:

```
TACL> RUN <your replicat>/in paramfile, name $xxxx, lib/
```

This decouples your Replicat from our licensed PRIVLIB and prevents errors.

If you are running the native form of Replicat, you may debug as usual.

Sample user exits

Two sample user exits are supplied with GoldenGate: DEMOXCOB (written in COBOL) and dDEMOXC (written in C). You can use these exits as skeletons for your own routines.

DEMOXCOB illustrates several applications of user exits. DEMOXCOB responds to Extract events and performs several tasks, including:

- Mapping data from Enscribe to SQL formats
- Writing a record to an attention log file under certain conditions
- Rejecting records with invalid codes

- Accumulating and outputting order totals
- Writing archive records when delete records are encountered

DEMOXC provides an example of how to write a user exit that responds to Replicat events. DEMOXC maps records from a source to a target layout and creates a summary transaction record for each delivered transaction.

Using GoldenGate macros

By using GoldenGate macros in parameter files you can easily configure and reuse parameters, commands, and functions. You can use macros for a variety of operations, including:

- Enable easier and more efficient building of parameters.
- Write once and use many times.
- Consolidate multiple statements.
- Eliminate redundant column specifications.
- Macros can invoke other macros.
- Create Macro libraries to share across parameter files.

GoldenGate macros work with Extract and Replicat parameter files

Creating a macro

Create a GoldenGate macro with the MACRO statement.

Syntax

```
MACRO #<macro name>
PARAMS (<p1>,<p2>...)
BEGIN
<macro body>
END;
```

Argument	Description
`MACRO #<macro name>`	Defines a GoldenGate macro. <macro name> must begin with the # character, as in #macro1. If the # macro character is used elsewhere in the parameter file, such as in a table name, you can change it to something else with the MACROCHAR parameter. See "Changing the macro character" on page 105. Macro names are not case-sensitive.
`PARAMS (<p1>,<p2>...)`	Optional. Used to describe parameters to the macro. Each parameter used in the macro must be declared in the PARAMS statement. See "Creating macro parameters" on page 105 for details about this option.
`BEGIN`	Indicates the beginning of the body of the macro. Must be specified before the macro body.

Argument	Description

`<macro body>`	Represents one or more statements to be used as parameter file input. <macro body> can include simple parameter statements, such as
	`COL1 = COL2`
	or more complex statements that include parameters, such as
	`COL1 = #val2`
	
	In addition, <macro body> may include invocations of other macros. For example:
	`#colmap(COL1, #sourcecol)`

`END`	Ends the macro definition.

Creating macro parameters

When you specify the optional PARAMS statement in a macro, the macro processor reads through the macro body looking for instances of the parameter names you defined in the PARAMS statement. For each occurrence of a parameter name, you must specify a corresponding value, which is substituted for the parameter name during invocation.

For example, to convert a proprietary date format, the following macro defines the #year, #month, and #day parameters.

```
MACRO #make_date
PARAMS (#year, #month, #day)
BEGIN
   @DATE(“YYYY-MM-DD”, “CC”, @IF(#year < 50, 20, 19),
         “YY”, #year, “MM”, #month, “DD”, #day)
END;
```

Parameter values are substituted within the macro body according to the following rules.

1. Parameter names must begin with the # macro character, such as #param1. (You can define a different macro character. See “Changing the macro character” on page 105.). When the macro is invoked, the invocation must include a parameter value for each parameter named in the PARAMS statement. Parameter names are not case-sensitive.

   When the macro processor encounters a parameter with the # prefix that is not in the parameter list, the processor determines whether or not it is an invocation of another macro. Invocations of other macros also begin with the # character, followed by parentheses enclosing a list of parameter values that are separated by commas.

2. Besides the leading # character, valid parameter characters are alphanumeric and can include the underscore character (_).

3. If a parameter name or macro is encountered within quotation marks, it is treated as text and ignored.

Changing the macro character

Anything in the parameter file that begins with the # macro character is assumed to be either a macro or macro parameter. This rule does not apply to text within quotation marks; quoted text is ignored.

If the macro character conflicts with a specification in the parameter file, such as table names that include the # character, you specify a different macro character with the MACROCHAR parameter. In the following example, $ is defined as the macro character, rather than #.

```
MACROCHAR $

MACRO $mymac
PARAMS ($p1)
BEGIN
   col = $p1
END;
```

The MACROCHAR can only be specified once, and must be specified before any macros are defined.

Invoking the macro

To invoke a macro, place the invocation statement in the parameter file at every place you want the process to occur.

Syntax `[<target> =] #<macro name> (<val1>, <val2>, ...)`

Argument	Description
`<target> =`	An optional target to which the results of the macro processing are assigned, such as: `DATECOL1 = #make_date(YR1, MO1, DAY1).`
`#<macro name>`	The name of the macro, such as #assign_date.

`(<val1>, <val2>...)`	The parameter values to be substituted inside the macro, such as #custdate (#year, #month, #day). If the optional PARAMS statement is omitted, the parenthesis are still required. See the section on invoking macros without parameters on page 106 for more information.
	Valid parameter values include plain text, quoted text, and invocations of other macros. Some examples of valid parameter values are:
	`my_col_1`
	`“your text here”`
	`#mycalc (col2, 100)`
	`#custdate (#year, #month, #day)`
	`#custdate (#getyyyy (#yy), #month, #day)`

Invoking a macro without parameters

If the macro does not specify parameters, the parameter value list is empty, but the parentheses are still required. For example:

```
#no_params_macro ()
```

Sample macros

This section shows you sample macros for implementing multiple uses of a statement and invoking another macro.

Implementing multiple uses of a statement

You can use macros to implement multiple uses of a statement, and eliminate the need for entering one statement several times.

The following example illustrates how mapping can be improved with a macro. In this example, a proprietary date format needs to be converted and the process is used several times. For such a scenario, you could implement a date format conversion in a macro similar to the following:

```
MACRO #make_date
PARAMS (#year, #month, #day)
BEGIN
@DATE(“YYYY-MM-DD”, “CC”, @IF(#year < 50, 20, 19),
“YY”, #year, “MM”, #month, “DD”, #day)
END;
```

To invoke the macro

1. Place the invocation statements at the appropriate location, similar to:

```
MAP $DATA.PROD.ACCOUNT, TARGET $DATA.BACK.ACCOUNT,
COLMAP (
    TARGCOL1 = SOURCECOL1,
    DATECOL1 = #make_date(YR1,MO1,DAY1),
    DATECOL2 = #make_date(YR2,MO2,DAY2)
    );
```

2. Upon invocation, the macro expands to:

```
MAP $DATA.PROD.ACCOUNT, TARGET $DATA.BACK.ACCOUNT,
COLMAP(
    TARGCOL1 = SOURCECOL1,
    DATECOL1 = @DATE(“YYYY-MM-DD”, “CC”, @IF(YR1 < 50, 20, 19),
               “YY”, YR1, “MM”, MO1, “DD”, DAY1)
    DATECOL2 = @DATE(“YYYY-MM-DD”, “CC”, @IF(YR2 < 50, 20, 19),
               “YY”, YR2, “MM”, MO2, “DD”, DAY2)
    );
```

Consolidating multiple commands

In addition, frequently invoked sets of commands can be specified in a macro, as in this example of the macro #option_defaults.

```
MACRO #option_defaults
BEGIN
  GETINSERTS
  GETUPDATES
  GETDELETES
  INSERTDELETES
END;
```

Invoking the macro:

```
#option_defaults ()
IGNOREUPDATES
MAP $DATA.PROD.TCUSTMER, TARGET $DATA.BACK.TCUSTMER;
```

expands to:

```
GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
IGNOREUPDATES
MAP $DATA.PROD.TCUSTMER, TARGET $DATA.BACK.TCUSTMER;
```

Invoking the macro:

```
#option_defaults ()
MAP $DATA.PROD.TCUSTORD, TARGET $DATA.BACK.TCUSTORD
```

expands to:

```
GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
MAP $DATA.PROD.TCUSTORD, TARGET $DATA.BACK.TCUSTORD;
```

Macro libraries

You can create libraries of macros to be included in different parameter files.

To create a macro library:

1. Create the macros using a text editor, saving them to a file name with the format $DATA.GGSMACR.FILENAME, where FILENAME is the name of the file.

   NOTE A macro library file can contain multiple macros.

2. Store your macro library files in $DATA.GGSMACR.
3. Specify the INCLUDE parameter in your parameter file to include the macro library.

Sample macro libraries

These samples show:

1. The macro library $DATA.GGSMACR.DATELIB that contains macros for date conversions.
2. The $DATA.GGSMACR.MAINLIB macro library containing the macro with multiple commands.
3. A sample parameter file calling a macro library that shows the include statement, and invocation statements for one of the macros from the library.

   This example is an Extract parameter file using the DATELIB macro library, and the #assign_date macro.

Figure 26 The $DATA.GGSMACR.DATELIB macro library

This example contains two macros, #make_date and #assign_date.

```
--
-- Date macro library
--
MACRO #make_date
PARAMS (#year, #month, #day)
BEGIN
@DATE("YYYY-MM-DD", "CC", @IF(#year < 50, 20, 19),
"YY", #year, "MM", #month, "DD", #day)
END;
MACRO #assign_date
PARAMS (#target_col, #year, #month, #day)
BEGIN
#target_col = #make_date (#year, #month, #day)
END;
```

Figure 27 The $DATA.GGSMACR.MAINLIB macro library

```
--
-- Main macro library
--
INCLUDE $DATA.GGSMACR.DATELIB
MACRO #option_defaults
BEGIN
GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
END;
```

Figure 28 Sample Extract parameter file

```
-- Parameter file for EXTRACT EXT1
--
INCLUDE $DATA.GGSMACR.DATELIB
EXTRACT EXT1
...
MAP $DATA.PROD.ACCOUNT, TARGET $DATA.BACK.ACCOUNT,
COLMAP (
TARGCOL1 = SOURCECOL1,
#assign_date(DATECOL1,YR1,MO1,DAY1),
#assign_date(DATECOL2,YR1,MO1,DAY1)
);
...
```

The parameter file processes the macro as follows:

- Notice that the INCLUDE statement pointing to DATELIB is specified at the beginning of the parameter file.
- The #assign_date macro is invoked when needed.

Suppressing report file listing

When including long, standard macro libraries, you may want to suppress listing each macro in the report file. Listing can be turned off and on by placing the LIST and NOLIST commands anywhere within the parameter file or within the included library.

For example, in the following, NOLIST suppresses listing each macro in hugelib. Specifying LIST after the INCLUDE statement restores normal listing to the report file.

```
NOLIST
include $DATA.GGSMACR.HUGELIB
LIST
EXTRACT EXT1
...
```

Tracing parameter expansion

You can trace macro expansion with the CMDTRACE parameter. When CMDTRACE is enabled, the macro processor displays macro expansion steps in the process's report file.

The syntax is:

Syntax `CMDTRACE [ON | OFF | DETAIL]`

Argument	Description
ON	Enables tracing.
OFF	Disables tracing. This is the default setting.

Example In the following example, tracing is enabled before #testmac is invoked, then disabled after the macro's execution.

```
EXTRACT EXT1
MACRO #testmac
BEGIN
COL1 = COL2,
COL3 = COL4
END;
...
CMDTRACE ON
MAP $DATA.TEST.TEST1, TARGET $DATA.TEST.TEST2,
COLMAP
(
#testmac
);
CMDTRACE OFF
....
```

Using OBEY files

With OBEY files, you can direct GoldenGate to parameters stored in a different file, then return processing to the current parameter file. OBEY files are useful for frequently used parameter statements, or parameters that are used by multiple parameter files.

Syntax

```
OBEY <file name>
```

To use an OBEY file:

1. Use the NonStop editor to create a file and enter the desired parameters.
2. Edit the file where you wish to place an OBEY statement.
3. Enter the OBEY command, specifying the name of the file as <file name>.

Example

```
TACL> RUN GGSCI
GGSCI> OBEY $DATA03.GGS.FINANCE
```

Using DEFINE names

ADD DEFINE lets you specify any value that is valid on HP NonStop to override GoldenGate defaults and allows user-defined names for:

- parameter file volumes and subvolumes
- report file volumes and subvolumes
- AUDCFG segment used by BASELIB
- GoldenGate environment prefix
- TABLE NAMES resolution
- referring to a trail name in GGSCI

Using ADD DEFINE

While GoldenGate recommends you use defaults whenever possible, you can define alternatives in the GLOBALS parameter file using ADD DEFINE. To use ADD DEFINE, you will need perform the following procedure:

1. Open your GLOBALS parameter file for editing.
2. Add the following syntax:

   ```
   TACL> ADD DEFINE =<object to rename>, class, file location, including
   NODE
   ```

   For example:

   ```
   TACL> ADD DEFINE =pmptrail, class MAP, file \PROD.$DATA3.acctdat.pf
   ```

3. Save the file and exit.

NOTE You must specify a fully qualified file name for each ADD DEFINE statement you include in your GLOBALS parameter file. Failure to specify the node may cause your process to ABEND.

Creating high pin processes

Use the PCREATE library to intercept the C runtime creation of new processes in order to create high pin processes.

NOTE The PCREATE intercept is only available for native mode on H06/J06.

Replicat

For Replicat PCREATE must be combined with the relinkable PRIVLIB to build a combined library that will include intercepts to create a high pin TACL.

The following example combines the PCREATE intercept object, PCREATEO, with the relinkable (R) native mode (N) PRIVLIB to create a new user library named PRIVLIBX.

```
eld -ul -o PRIVLIBX PRIVLIRN PCREATEO -set interpose_user_library on
FUP LICENSE PRIVLIBX
eld -change libname $DATA.GGS1000.PRIVLIBX REPLICAT
```

In the last step the new PRIVLIBX is assigned as Replicat's library. The library name must be fully qualified as shown in the example.

Extract and GGSCI

Extract and GGSCI can use a combined library, such as created in the above example, or PCREATE can be linked into a user library, such as the PCREATEL in the example below.

```
eld -ul -o PCREATEL PCREATEO -set interpose_user_library_on
```

TACL DEFINE

The DEFINE for TACL, =GGS_TACL_PROGRAM, can be entered in GLOBALS if it is to be the same for all Extract, Replicat, and GGSCI programs for that GoldenGate instance. To make the define group specific, include it in the Extract or Replicat parameter file. Alternately it can also be added to TACLLOC or to TACLCSTM..

The following example DEFINE assumes you first FUP DUP $SYSTEM.SYS<nn>.TACL to $SYSTEM.SYS<nn>.TACLHP , turn HighPin ON, then add the DEFINE.

```
ADD DEFINE =GGS_TACL_PROGRAM, CLASS MAP, FILE $SYSTEM.SYS<nn>.TACLHP
```

CHAPTER 9

Integrating Data

You can integrate just the data you need into your target using parameters, clauses, column mapping, and functions. This chapter guides you through each technique.

Selecting records

You can select specific records to extract or replicate using the FILTER and WHERE clauses of the TABLE or MAP parameters. FILTER is the more powerful tool, letting you filter records on a variety of criteria. You may specify multiple filters in one FILE, TABLE, or MAP statement. However, WHERE is a quick, simple way to select a record that matches a single criteria. You may only have one WHERE clause per statement.

Selecting records with FILTER

Use the FILTER clause of FILE, TABLE, or MAP to select specific records within a file or table for Extract or Replicat. FILTER uses the GoldenGate field conversion functions to evaluate whether or not to process a record. For example, the following statement extracts records in which the price multiplied by the amount exceeds 10000:

```
TABLE $DATA.MASTER.CUSTOMER, FILTER
((PRODUCT_PRICE*PRODUCT_AMOUNT)>10000);
```

In another example, the following extracts records containing a string JOE:

```
TABLE $DATA.MASTER.CUSTOMER, FILTER (@STRFIND(NAME, "JOE")>0);
```

Selecting records with WHERE

Use the WHERE clause in TABLE or MAP to select specific records within a table to be extracted or replicated.

The WHERE clause consists of the following elements and must be enclosed in parentheses.

Element	Example
Columns from the row	`PRODUCT_AMT`
Numeric values	`-123, 5500.123`
Literal strings enclosed in quotes	`"AUTO", "Ca"`

Element	Example
Column tests	`@NULL,@PRESENT,@ABSENT` (column is null, present or absent in the record)
Comparison operators	`=, <>, >, <, >=, <=`
Conjunctive operators	`AND, OR`
Grouping parentheses	open and close parentheses () for logical grouping

Arithmetic operators and floating point data types are not supported. To perform more complex selection conditions, use FILTER.

Comparing fields

Ensure that the variable and value you specify in a comparison match appropriately. Compare:

- Characters with literal strings.
- Numeric fields with numeric values, which can include a sign and decimal point.
- SQL datetime types to literal strings, using the format in which the field is retrieved by a program.

Compressed update considerations

When a compressed update record is encountered for a table, only part of the record image is available for the condition evaluation. By default, when a column required by the condition evaluation is missing, the record is ignored and output to the discard file, and a warning is issued.

- Use only columns that appear in the primary key of the record, since key fields are always present in compressed records.
- Test for a column's presence first, then for the column's value.

To test for a column's presence, use the following syntax:

```
<field> [= | <>] [@PRESENT | @ABSENT]
```

The following example returns all records when the AMOUNT field is over 10000 and does not cause a record to be discarded when AMOUNT is absent.

```
WHERE (AMOUNT = @PRESENT AND AMOUNT > 10000)
```

Testing for NULL values

Evaluate SQL columns for NULL values with the @NULL clause.

The following test returns TRUE if the column is NULL, and FALSE for all other cases (including a column missing from the record).

```
WHERE (AMOUNT = @NULL)
```

The following test returns TRUE only if the column is present in the record and not NULL.

```
WHERE (AMOUNT = @PRESENT AND AMOUNT <> @NULL)
```

Column mapping

GoldenGate provides the capability to transform data between two dissimilarly structured database tables or files. These features are implemented with the COLMAP clause in the TABLE or MAP parameters described in this chapter.

Mapping between different database structures

Using GoldenGate, you can transform data to accommodate differences in source and target database structures.

For example:

- The source is a NonStop Enscribe file (ACCTFL), while the target is a SQL table (ACCTTAB).
- 75 fields exist in ACCTFL, while ACCTTAB contains only nine columns.
- Five columns in ACCTTAB have corresponding field names in the ACCTFL (ADDRESS, CITY, STATE, ZIPCODE, SOCIAL_SECURITY_NO).
- A ten digit phone number field in ACCTFL corresponds to separate area code, prefix, and phone number columns in ACCTTAB.
- A date column in ACCTTAB is computed from year, month and day fields in ACCTFL.

In this scenario, you can design a column map in a Replicat parameter file MAP statement on NonStop. For example:

```
MAP $DATA.MASTER.ACCTFL, DEF ACCOUNT-REC,
TARGET $DATA.MASTER.ACCTTAB,
COLMAP (
   USEDEFAULTS,
   NAME = CUST-NAME,
   TRANSACTION_DATE = @DATE ("YYYY-MM-DD",
      "YY", TRDATE.YEAR,
      "MM", TRDATE.MONTH,
      "DD", TRDATE.DAY),
   AREA_CODE = @STREXT (PHONE, 1, 3),
   PHONE_PREFIX = @STREXT (PHONE, 4, 6),
   PHONE_NUMBER = @STREXT (PHONE, 7, 10)
   );
```

This statement is composed of the following elements:

1. The source file (ACCTFL) and corresponding DDL definition for ACCOUNT-REC.
2. The target table name (ACCTTAB). No definition is required for the SQL table since it is retrieved automatically from a catalog.
3. The COLMAP parameter.
4. USEDEFAULTS, which directs Replicat to move all fields in ACCTFL that have matching columns in ACCTTAB into the ACCTTAB table. Data translation between different data types is automatic.
5. An explicit assignment of the CUST-NAME field to the NAME column. This is required because the names are different.
6. A date calculation for TRANSACTION_DATE based on three fields in ACCTFL.

7. Extracting parts of PHONE-NO into AREA_CODE, PHONE_PREFIX and PHONE_NUMBER.

Data type conversions

Numeric fields are converted from one type and scale to match the type and scale of the target. If the scale of the source is larger than that of the target, the number is truncated on the right. If the target scale is larger than the source, the number is padded with zeros.

Varchar and character columns can accept other character, varchar, group, and datetime columns, or string literals enclosed in quotes. If the target character column is smaller than that of the source, the character column is truncated on the right.

Datetime fields can accept datetime and character columns, as well as string literals. If you attempt to map a character into a datetime column, make sure it conforms to the GoldenGate external SQL format (YYYY-MM-DD:HH:MI:SS.FFFFFF). Required precision varies according to data type and target platform. Datetime columns are truncated on the right as necessary. If the source column is not as long as the target, the column is extended on the right with the values for the current date and time.

GoldenGate user tokens

GoldenGate user tokens allow you to capture data and values for use in data integration. User tokens are composed of alphanumeric data from your source system, database, transactions, and/or records. They can also transfer values into other user tokens generated by queries, procedures, or other called functions.

NOTE The user token area is limited to 2000 bytes of information. Token names, data length, and the data itself is all used to calculate the user token area size.

User tokens are stored in each record's trail header, and retrieved by the appropriate GoldenGate component. The following tables outline types of data that appear in user tokens.

Table 5 Sample environmental data for user tokens

Environmental Detail	Description
GROUPNAME	Extract or Replicat group name.
HOSTNAME	Host name running the Extract or Replicat.
OSUSERNAME	The user name that started Extract or Replicat.

Table 6 Sample header details and their description

Header Detail	Description
BEFOREAFTERINDICATOR	Before/after indicator
COMMITTIMESTAMP	Commit timestamp
LOGPOSITION	Log position

Table 6 Sample header details and their description

Header Detail	Description
LOGRBA	Log RBA
TABLENAME	Table name
OPTYPE	Operation type
RECORDLENGTH	Record length
TRANSACTIONINDICATOR	Transaction indicator

Populating user tokens in the trail header

To populate user tokens in the trail header, you must include a TOKEN clause on the FILE or TABLE parameter in the Extract parameter file. To do so, complete the following procedure:

1. Edit the Extract parameter file.

   ```
   GGSCI> TEDIT PARAMS EXTDEMO
   ```

2. Specify a table name

   ```
   TABLE $DATA.MASTER.PRODUCT,
   ```

3. Enter the desired tokens. The @GETENV function, quotation marks and comma delimiter are required.

   ```
   TOKENS
      (
      TKN-GROUP-NAME   =@GETENV (“GGENVIRONMENT”, “GROUPNAME”),
      TKN-HOST-NAME    =@GETENV (“GGENVIRONMENT”, “HOSTNAME”),
      TKN-OS-USER      =@GETENV (“GGENVIRONMENT”, “OSUSERNAME”),
      TKN-BA           =@GETENV (“GGHEADER”, “BEFOREAFTERINDICATOR”),
      TKN-COMMIT-TS    =@GETENV (“GGHEADER”, “COMMITTIMESTAMP”),
      TKN-LOG-POSITION =@GETENV (“GGHEADER”, “LOGPOSITION”),
      TKN-LOG-RBA      =@GETENV (“GGHEADER”, “LOGRBA”),
      TKN-TABLE        =@GETENV (“GGHEADER”, “TABLENAME”),
      TKN-OP-TYPE      =@GETENV (“GGHEADER”, “OPTYPE”),
      TKN-REC-LEN      =@GETENV (“GGHEADER”, “RECORDLENGTH”),
      TKN-TRNS-IND     =@GETENV (“GGHEADER”, “TRANSACTION INDICATOR”),
      );
   ```

4. Exit the parameter file.

Retrieving values

To retrieve values, you must include a MAP parameter and a COLMAP clause in the Replicat parameter file, then use the @TOKEN function to specify the values to retrieve.

```
MAP $DATA.MASTER.PRODUCT, TARGET $DATA.MASTER.PRODUCT_CHANGES,
COLMAP (USEDEFAULTS,
SOURCE_GROUP        =@TOKEN (“TKN-GROUP-NAME”),
SOURCE_HOST         =@TOKEN (“TKN-HOST-NAME”),
SOURCE_USER         =@TOKEN (“TKN-OS-USER”),
BEFORE_AFTER_IND    =@TOKEN (“TKN-BA”),
TIMESTAMP           =@TOKEN (“TKN-COMMIT-TS”),
SOURCE_TABLE        =@TOKEN (“TKN-TABLE”),
IO_TYPE             =@TOKEN (“TKN-OP-TYPE”));
```

The @TOKEN function requires quotation marks.

Default mapping

When you specify COLMAP USEDEFAULTS, Extract maps columns in the source table to columns in the target with the same name. At startup, Extract outputs column names that match and will map to each other.

The USEDEFAULTS parameter allows matching columns to be mapped, plus additional columns. This can be useful when the source and target definitions are similar but not identical.

If you set up global column mapping rules with COLMATCH parameters, you can map columns with different names to each other using default mapping. See the Extract and Replicat COLMATCH parameter for more details.

When unspecified or no match is found in a default map, a target field defaults to one of the following:

Column	Value
Numeric	Zero
Character or varchar	Spaces
Datetime	Current date and time
Columns that can take a NULL value	NULL

If the target table contains names corresponding to the transactional columns described above, the special column values are mapped to the target record format.

Mapping examples

The following is the source Enscribe DDL for the examples in this section.

```
RECORD PRODUCT-REC.
FILE IS PRODDAT KEY-SEQUENCED AUDIT.
05 PROD-KEY.
   10 CODE1 PIC X(2).
   10 CODE2 PIC 9(2).
05 PROD-INDEX1.
   10 PRICE PIC 9(7)V9(2) COMP.
   10 CODE1 PIC X(2).
   10 CODE2 PIC 9(2).
05 PROD-INDEX2.
   10 INVENTORY PIC 9(5).
   10 CODE1 PIC X(2).
   10 CODE2 PIC 9(2).
05 DESC PIC X(40).
KEY IS PROD-KEY.
END.
```

The following is the target SQL DDL for the examples in this section.

```
Target SQL DDL
CREATE TABLE PRODTAB
(
CODE CHAR(4) NOT NULL
, PRICE NUMERIC (8,2) NOT NULL
, INVENTORY DECIMAL (6)
, MANAGER CHAR (20) NOT NULL
, DESC VARCHAR (30)
, UPDATE_TIME DATETIME YEAR TO SECOND NOT NULL
, PRIMARY KEY (CODE)
);
```

Legal column mapping

Note that one can move a group level (PROD-KEY) to a character field. This is feasible since CODE2 is a DISPLAY field, not a COMP. Also, the user does not have to qualify PRICE, INVENTORY or DESC since they are all unique in the source definition. UPDATE_TIME will default to the time at which EXTRACT processes the record. PRICE may be truncated since it has one more significant digit in the source field than in the target.

```
FILE $DAT11.OLDAPP.PRODFL,
DEF PRODUCT-REC,
TARGET $DATA6.NEWAPP.PRODTAB,
COLMAP
(CODE = PROD-KEY,
PRICE = PROD-INDEX1.PRICE,
INVENTORY = INVENTORY,
MANAGER = “Jack Smith”,
DESC = DESC);
```

Dangerous mapping if AUDITCOMPRESS used on source file

Since this mapping takes the primary key value from a non-primary key source, it discards the result whenever a source record is updated without updating the price. In the following example, even if AUDITCOMPRESS is used, updates can be delivered since the primary key is always present.

```
FILE $DAT11.OLDAPP.PRODFL,
DEF PRODUCT-REC,
TARGET $DATA6.NEWAPP.PRODTAB,
COLMAP
(CODE = PROD-INDEX1.CD1,
PRICE = PROD-INDEX1.PRICE,
INVENTORY = INVENTORY,
MANAGER = “Unknown”,
DESC = DESC);
```

Using constants, taking default values.

This mapping sets PRICE to zero and Manager to spaces since they are not null fields, and sets INVENTORY and DESC to NULL since they can take null values.

```
TABLE $DAT11.OLDAPP.PRODFL,
DEF PRODUCT-REC,
TARGET $DATA6.NEWAPP.PRODTAB,
COLMAP
(CODE = PROD-KEY,
UPDATE_TIME = “2006-01-01:08:00:00”);
```

Field conversion functions

Using field conversion functions, you can manipulate numbers, strings and source column or field values into the appropriate format for target columns.

See the *Reference Guide* for more information about column conversion functions.

Function arguments

Column conversion functions can take one or more of the following parameters.

Parameter	Example
A numeric constant	`123`
A string constant	`"ABCD"`
A column or field from the source table or file	`PHONE-NO.AREA-CODE or COLUMN_3`
An arithmetic expression	`COL2 * 100`
A comparison expression	`COL3 > 100 AND COL4 > 0`

Parameter	Example
A field conversion function	its own parameters

NOTE Argument checking at run-time is not always strict and errors in argument passing are sometimes not detected until records are processed.

Arithmetic expressions

Arithmetic expressions can be combinations of the following elements.

- Numbers
- Columns that contain numbers
- Functions that return numbers
- Arithmetic operators: + (plus), - (minus), * (multiply), / (divide), \ (remainder)
- Comparison operators: > (greater than), >= (greater than or equal), < (less than), <= (less than or equal), = (equal), <> (not equal)
- Parentheses (for grouping results in the expression)
- Conjunction operators: AND, OR

To return the result of an arithmetic expression to a column, use the COMPUTE function.

The COMPUTE function is not required when an expression is passed as an argument, as in @STRNUM (AMOUNT1 + AMOUNT2, RIGHT).

@STRNUM (@COMPUTE(AMOUNT1 + AMOUNT2), RIGHT) would return the same result.

Arithmetic results derived from comparisons are zero (indicating FALSE) or non-zero (indicating TRUE).

When conjunction operators are involved in an expression, only the necessary part of the expression is evaluated. Once a statement is FALSE, the rest of the expression is ignored. This can be valuable when evaluating fields that may be missing or null.

For example, assume the value of COL1 is 25 and the value of COL2 is 10:

```
@COMPUTE (COL1 > 0 AND COL2 < 3) returns 0
@COMPUTE (COL1 < 0 AND COL2 < 3) returns 0 (and COL2 < 3 is never
evaluated)
@COMPUTE ((COL1 + COL2)/5) returns 7
```

See the *Reference Guide* for details about the functions.

Null, invalid, and missing columns and fields

One problem encountered when calculating column values is that some data may be missing from the expression. More specifically, source columns or fields may have one of the status conditions explained in the following table.

Table 7

Column Status	Description
Missing	Frequently, data is missing in compressed update records. Compressed update records contain only those source columns that changed, plus the key of the source file or table.
Null	A source column may contain a null value, which makes a calculation difficult.
Invalid	The source data is invalid.

When one of these conditions occurs, by default the condition is returned as the result of the function.

For example, if BALANCE is 1000, but AMOUNT is NULL, the following expression returns NULL.

```
NEW_BALANCE = @COMPUTE (BALANCE + AMOUNT)
```

As another example, the AMOUNT field is defined as PIC 9(5)V99 in an Enscribe record definition, but contains spaces. In that case, the above expression returns INVALID, and the record is discarded.

If AMOUNT, but not BALANCE, is present in the update record, the field is not mapped.

Overriding exceptional conditions

The IF, COLSTAT and COLTEST functions recognize null, invalid, or missing columns and can compute alternative values.

For example:

```
NEW_BALANCE = @IF (@COLTEST (BALANCE, NULL, INVALID) OR
                @COLTEST (AMOUNT, NULL, INVALID),
                @COLSTAT (NULL),
                BALANCE + AMOUNT)
```

returns one of the following:

- NULL when BALANCE or AMOUNT is NULL or INVALID,
- MISSING when either column is missing
- The sum of the columns.

CHAPTER 10
Managing and Monitoring GoldenGate

Once you have initialized your source and target and started change synchronization, you are ready to manage your GoldenGate environment. This chapter introduces a variety of tools and techniques for these administrative tasks.

Managing tasks

Tasks are processes that are special runs, such as a one-time data synchronization, or direct file extraction. Tasks are useful in managing GoldenGate, because they allow you to load data that may have been missed due to a variety of system errors. You can define a task with the GGSCI commands:

```
GGSCI> ADD EXTRACT <group name>, SOURCEISTABLE
GGSCI> ADD REPLICAT <group name>, SPECIALRUN
```

When you define a task, you must include the task type parameter in the parameter file. For the Extract parameter file, include SOURCEISTABLE or SOURCEISFILE. For the Replicat parameter file include SPECIALRUN.

Manager can purge tasks. To purge tasks enter parameters such as:

```
PURGEOLDTASKS EXTRACT <wildcard spec>, AFTER <n> HOURS, USESTOPSTATUS
PURGEOLDTASKS REPLICAT <wildcard spec>, AFTER <n> DAYS, USESTOPSTATUS
PURGEOLDTASKS ER <wildcard spec>, AFTER <n> HOURS, USESTOPSTATUS
```

Getting information on tasks

You can retrieve information about a task using the INFO and STATUS commands with the TASKS or ALLPROCESSES options:

```
GGSCI> INFO EXTRACT *, TASKS
GGSCI> INFO REPLICAT *, ALLPROCESSES
GGSCI> STATUS ER *, ALLPROCESSES
```

TASKS reports on SPECIALRUN or SOURCEISFILE tasks. ALLPROCESSES reports on all processes.

Managing tasks using the process name

Tasks defined with SPECIALRUN, SOURCEISFILE, or SOURCEISTABLE do not require a group name. Even without this name, it is possible to communicate with these running tasks by using the SEND PROCESS command. The syntax for this uses the process name instead of a group name as shown below.

```
GGSCI> SEND PROCESS <process name> {<text> | WAKE | BREAK}
```

The <text> option can be any one of the subset of GGSCI commands that are recognized by the receiving process.

See *Reference Guide* for details about the SEND PROCESS options.

Managing GoldenGate trails

GoldenGate trails can be managed by allocating optimal storage for the trail files and setting parameters for cleaning up trail files that are no longer needed.

Initial allocating of storage for trails

To prevent trail activity from interfering with business applications, use a separate disk managed by a disk process different than that of the application.

To ensure there is enough disk space for the trail files, follow these guidelines:

- For trails on the source system, there should be enough space to handle data accumulation if the network connection fails. In a failure, reading from a trail terminates but the primary Extract group reading from logs or audit file continues extracting data. It is not good practice to stop the primary Extract group to prevent further accumulation. The logs could recycle or the audit files could be off-loaded.
- For trails on the target system, data will accumulate because data is extracted and transferred across the network faster than it can be applied to the target database.

To estimate the required trail space

1. Estimate the longest time that you think the network can be unavailable.
2. Estimate how much transaction log volume you generate in one hour.
3. Use the following formula:

```
trail disk space =
<transaction log volume in 1 hour> x <number of hours down> x .4
```

NOTE The equation uses a multiplier of 40 percent because GoldenGate estimates that only 40 percent of the data in the transaction logs is written to the trail.

A more exact estimate can be derived by either:

1. Configuring Extract and allowing it to run for a set time period, such as an hour, to determine the growth. This growth factor can then be applied to the maximum down time.
2. Using MEASFLS and MEASRPT to collect and report on statistics over a full business cycle and using this data to determine the volume over the maximum down time.

Plan to store enough data to withstand the longest anticipated outage possible because you will need to resynchronize the source and target data should the outage outlast the disk capacity.

Ongoing trail management

GoldenGate provides options that allow you to manage your trails in two ways.

- Based on the number and size of the files.

  The MEGABYTES, MAXFILES, and EXTENTS options specify how large each trail file may become, and how many files may exist before Extract stops with an error.

- With the PURGEOLDEXTRACTS parameter.

  This allows you to purge old extracted data you no longer need. This can be based on rules you set up.

  - The MINKEEPHOURS, MINKEEPDAYS options set the time to keep files. MINKEEPFILES sets the minimum number of files to keep.
  - In the Manager only, the USECHECKPOINTS option uses checkpoints to determine whether processing is complete. You can also set the CHECKMINUTES parameter to control how often the process checks the parameters to determine if anything needs to be purged.

Setting the size of the trail

Two options for managing trail size are MEGABYES and MAXFILES. MEGABYTES lets you specify how large your trail file may get before your data rolls to another trail file. It is useful if you wish to equally distribute data between your files. The default size is 134 megabytes and the largest size supported is two gigabytes. MAXFILES lets you specify the number of trail files GoldenGate may start. The default is 100 files. Allowing multiple files lets data rollover when one file is full, which prevents errors. The syntax for using MEGABYTES and MAXFILES is:

```
GGSCI> ADD EXTTRAIL <trail name>, EXTRACT <group name>, MEGABYTES <nn>,
MAXFILES [n]
```

Trails that either reside on the local node or on a node that is connected by Expand are considered local for NonStop. For these trails, size can also be controlled by setting the files' primary, secondary and maximum number of extents. The syntax for this is:

```
GGSCI> ADD EXTTRAIL <trail name>, EXTRACT <group name>
[, EXTENTS (<primary, secondary, max>)]
```

The defaults for EXTENTS are (64, 128, 512).

Figure 29 From GGSCI, an INFO of the trail will show the current trail settings.

```
GGSCI> INFO EXTTRAIL GGSDAT.ET
Extract file: \NY.$DATA04.GGSDAT.ET
       Extract group: EXTSQL
               Owner: 150,110
            Security: NUNU
       Current seqno: 0
         Current rba: 2280
      Primary extent: 64
    Secondary extent: 128
         Max extents: 512
           Max files: 100
```

Setting the PURGEOLDEXTRACTS rules

In the Manger

Options for purging trails can be set with the PURGEOLDEXTRACTS in the Manager's parameter file.

- Use USECHECKPOINTS to purge when all processes are finished with a file as indicated by checkpoints. This is the default, but it can be turned off with the NOUSECHECKPOINTS option.
- MINKEEPHOURS or MINKEEPDAYS to keep files n hours or days. MINKEEPFILES to keep at least n files including the active file. The default is 1.

Only *one* of the three MINKEEP options should be set. If more than one is entered the system will select one based on the following:

- If both MINKEEPHOURS and MINKEEPDAYS are specified, only the last setting will be used and the other will be ignored.
- If both MINKEEP<time> and MINKEEPFILES are specified MINKEEP<time> will be used and MINKEEPFILES will be ignored.

In Extract or Replicat

There are no options available for the Extract and Replicat PURGEOLDEXTRACTS parameter. In this case the trail is purged as soon as the process moves to the next trail.

Manager purge trail processing

When the Manager reaches CHECKMINUTES, if PURGEOLDEXTRACTS is set in the Manager parameter file, the purge rules are evaluated as explained below.

1. USECHECKPOINTS only

   If there are no minimum rules set with the USECHECKPOINTS option, MINKEEPFILES defaults to 1. If checkpoints indicate that a trail file has been processed, it will be purged unless it would fall below this one file minimum.

2. USECHECKPOINTS with MINKEEP rules

   If checkpoints indicate that a trail file has been processed, it will be purged unless doing so would violate the applicable MINKEEP<time> or MINKEEPFILES rules. Refer to "In the Manger" on page 126 for information on setting these PURGEOLDEXTRACTS minimum rules.

3. NOUSECHECKPOINTS only

   If there are no minimum rules and checkpoints are not to be considered, the the file will purged unless doing so will violate the default MINKEEPFILES of 1.

4. NOUSECHECKPOINTS with MINKEEP rules

   The file will be purged unless doing so will violate applicable MINKEEP<time> or MINKEEPFILES rules. Refer to "In the Manger" on page 126 for information on setting these PURGEOLDEXTRACTS minimum rules.

Purge processing examples

Example Trail files AA000000, AA000001, and AA000002 exist. The Replicat has been down for four hours and has not completed processing any of the files.

The Manager parameters include:

```
PURGEOLDEXTRACTS $DATA1.DB.AA*, USECHECKPOINTS, MINKEEPHOURS 2
```

Result: The time the unaccessed files must be retained has been exceeded. No files will be purged, however, because checkpoints indicate that the files have not been fully processed by Replicat.

Example Trail files AA000000, AA000001, and AA000002 exist. The Replicat has been down for four hours and has not completed processing.

The Manager parameters include:

```
PURGEOLDEXTRACTS $DATA1.DB.AA*, NOUSECHECKPOINTS, MINKEEPHOURS 2
```

Result: All trail files will be purged since the minimums have been met.

Example The following is an example of why only one of the MINKEEP options should be set.

Replicat and Extract have completed processing. There has been no access to the trail files for the last five hours. Trail files AA000000, AA000001, and AA000002 exist.

The Manager parameters include:

```
PURGEOLDEXTRACTS $DATA1.DB.AA*, USECHECKPOINTS, MINKEEPHOURS 4, MINKEEPFILES 4
```

Result: USECHECKPOINTS requirements have been met so the minimum rules will be considered when deciding whether to purge AA000002.

There will only be two files if AA000002 is purged, which will violate the MINKEEPFILES parameter. Since both MINKEEPFILES and MINKEEPHOURS have been entered, however, MINKEEPFILES is ignored. The file will be purged because it has not been modified for 5 hours, which meets the MINKEEPHOURS requirement of 4 hours.

The Manager process determines which files to purge based on the Extract processes configured on the local system. If at least one Extract process reads the trail file, Manager applies the specified rules.

For further information regarding PURGEOLDEXTRACTS and ADD EXTTRAIL options, see the *Reference Guide*.

Recommendations for managing trail purges

Consider the following recommendations for managing GoldenGate trails.

- For setting the purge rules, it is recommended that:
  - PURGEOLDEXTRACTS should be specified in the Manager because this allows you to to manage your trails from a single location.
  - Trail files should be purged by Extract or Replicat only when a single process is processing the trail, such as a data pump. Manager should be used to purge trail files that are being processed by both Extract and Replicat.

    - Using USECHECKPOINTS helps to ensure that the checkpoints of both Extract and Replicat are considered, which reduces the chance of data loss.
    - Using PURGEOLDEXTRACTS in Extract or Replicat can remove trails still needed by the Coordinator. Therefore, systems that use Coordinator should specify PURGEOLDEXTRACTS in the Manager to properly manage the Coordinator checkpoints.

- The rules should be assigned to the process that resides where the trail needs to be cleaned.

  For example, if there are three nodes: `\A` where the Extract is running and extracting the data; `\B` where a subset of the data is replicated and `\C` where another part of the data is replicated, it is the Manager on `\A` that should be assigned the parameters that define how to manage the trails.

  For USECHECKPOINTS, this Manager will need to know the location of the checkpoint files on `\B` and `\C`, but this can be done with REMOTECHKPT as shown below.

  ```
  GGSCI> ADD REMOTECHKPT \<node>.$<volume>.<subvolume>.REPCTXT
  ```

Managing log trails

Unlike trails that are created externally, if a trail created by Logger runs out of space there are no audit records to be reprocessed once the problem is fixed. This can cause data loss, so it is important to have adequate space available for log trails. The following steps help to do this.

- Include adequate trail space when the Logger process is added.

  Log trail files are pre-allocated during ADD LOGGER so this ensures that the space is available before the logging process begins.

- Monitor and adjust the trail space as needed.

  During processing, the number and size of each log trail can be adjusted as needed by editing the Logger parameter file with the command EDIT PARAM LOGPARM. Then the number of files can be increased or decreased by changing the NUMFILES option, and the size of each trail file can be adjusted by changing the MEGABYTES or EXTENTS. The changes will take effect when an ALTER LOGGER command is issued.

- Monitor the impact of the trail space on your system.

  If the next log trail file is not available when it is time to rollover, Manager will create it. This helps to ensure that data will not be lost, but it also means that you may have more log trail files than specified in the NUMFILES of LOGPARM.

Monitoring processing

You can monitor the state of GoldenGate processing with the following tools.

Tool	Description
The event log	The event log, named LOGGGS, shows processing events, messages, errors, and warnings generated by GoldenGate. You can view the event log from GGSCI using the VIEW GGSEVT command.

Tool	Description
Process reports	GoldenGate generates a report for Extract, Replicat, and Manager at the end of each run. The report provides information about runtime parameters and statistics, including process events and operations that were performed. The name of a process report is either Manager for the Manager process, or for Extract and Replicat, is the same as the group name. For example, a report for an Extract group EXTORD would have a report named GGSRPT.EXTORD. You can view the process report, save and store it, and use it for generating runtime statistics.
Record counts	You can produce record counts at designated times during processing by using the REPORTCOUNT parameter in the Extract and Replicat parameter files. Results are printed to the process report file and to the screen.
The discard file	The discard file logs information about operations that failed. To generate discard files, use the DISCARDFILE parameter in the Extract or Replicat parameter file. To control how discard files are generated, use the DISCARDROLLOVER parameter in the Extract or Replicat parameter file. The parameter has provisions for specifying when new files are created.
GGSCI INFO commands	Using the GGSCI commands INFO, SEND, STATUS, and STATS, you can retrieve information on GoldenGate processes.

Each of these tools is discussed in greater detail below.

Error handling

There are a number of error handling parameters available in GoldenGate. In addition, GoldenGate provides error-handling options for Replicat and TCP/IP processing.

Error handling parameters

Error handling parameters let you insert missing records, prevent duplicate records from being loaded. For a complete list, see the *Reference Guide*.

Handling Replicat errors

To control the way that Replicat responds to errors, use the REPERROR parameter in the Replicat parameter file. This parameter handles most errors in a default fashion (for example, to cease processing), and you can specify REPERROR options to handle other errors in a specific manner, or to ignore them altogether.

REPERROR provides the following options:

Option	Description
ABEND	Roll back the Replicat transaction and terminate processing. This is the default.
DISCARD	Log the error to the discard file but continue processing the transaction and subsequent transactions.
EXCEPTION	Treat the error as an exception. To handle an exception, create an entry with the MAP parameter that executes after the error. For example, you can map a failed update statement to an exceptions table dedicated to missing updates.
IGNORE	Ignore the error.
RETRYOP	Retry the operation. Use the MAXRETRIES argument with RETRYOP to specify the number of times to retry an operation. To control the interval between retries, use the RETRYDELAY parameter.
TRANSABORT	Abort the current target transaction and then retry it.

TCP/IP error handling

The TCPERRS file in the GoldenGate directory contains preset TCP/IP errors and instructions for how GoldenGate generally reacts to them. If a response is not explicitly defined in this file, GoldenGate responds to TCP/IP errors by exiting.

NOTE The Manager process is an exception. When Manager has an IP error it retries every 60 seconds and does not abend. It does not use the TCPERRS file to determine the number of retries or the delay.

Figure 30 The following is an example of the TCPERRS file.

```
#
# TCP/IP error handling parameters
# Default error response is abend
#
# error        Response            Delay (csecs)  Max Retries
ECONNABORTED  RETRY               1000          10
ECONNREFUSED  RETRY               1000          12
ECONNRESET    RETRY               500           10
ENETDOWN      RETRY               3000          50
ENETRESET     RETRY               1000          10
ENOBUFS       RETRY               100           60
ENOTCONN      RETRY               100           10
EPIPE         RETRY               500           10
ESHUTDOWN     RETRY               1000          10
ETIMEDOUT     RETRY               1000          10
NODYNPORTS    RETRY               100           10
```

Altering TCP/IP error handling parameters

To alter the instructions or add instructions for new errors, open the file in a text editor and change any of the values in the following columns:

- Error column: Specifies a TCP/IP error for which you are defining a response.
- Response column: Controls whether or not GoldenGate tries to connect again after the defined error.
- Delay column: Controls how long GoldenGate waits before attempting to connect again.
- Max Retries column: Controls the number of times that GoldenGate attempts to connect again before aborting.

See the *Reference Guide* for details about the TCP/IP error messages, their causes, effects, and recovery.

Using the discard file

The discard file logs information about operations that failed. To generate discard files, use the DISCARDFILE parameter in the Extract and Replicat parameter files.

Extract discard records have a header and a data portion; the discard file is entry-sequenced. Replicat produces discard records in an external, easy-to-understand format.

Full record images are provided when IO-TYPE is Delete, Insert or Update. Each record has the same format as if retrieved from a program reading the original file or table directly. For SQL tables, datetime fields, nulls and other fields are output exactly as a program would SELECT them into an application buffer. Even though datetime fields are represented internally as an eight byte timestamp, their external form can be up to 26 bytes expressed as a string. Enscribe records are retrieved as they exist in the original file.

Full record images are output unless the original file has the AUDITCOMPRESS attribute set to ON. When AUDITCOMPRESS is ON, compressed update records are generated whenever the original file receives an update operation. (A full image can be retrieved by Extract using the FETCHCOMPS parameter.)

When the operation type is Insert or Update, the image is the contents of the record *after* the change is made. When the operation type is Delete, the image is the contents of the record *before* the change.

To control how discard files are generated, use the DISCARDROLLOVER parameter in the Extract or Replicat parameter file. The parameter has provisions for specifying when new files are created.

Conflict detection with SQLEXEC

SQLEXEC works on SQL for NonStop databases to call queries you specify. This lets you leverage application rules to resolve conflicts between incoming source records. To do this, add a SQLEXEC parameter in a MAP statement in the Replicat parameter file.

Some applications that support distributed processing provide their own methods for handling conflicts. Such methods include IP persistent routers or application privileges that prevent multiple users from modifying the same data. These rules can be combined with the use of SQLEXEC procedures.

A SQLEXEC example

The following is an example of basic conflict detection based on a timestamp. It raises an exception based on a user-defined error number passed to Replicat, and a SQL query to check the value of the TIMESTAMP column. If the timestamp is newer than the one on the existing record, the update is executed. Otherwise, the operation is ignored (because the newer timestamp is considered more valid) and a message is generated to the Replicat process report.

Figure 31 REPLICAT Parameter File

```
REPERROR (9999, EXCEPTION)
MAP $DATA.MASTER.SRC, TARGET $DATA.MASTER.TAR,
SQLEXEC (ID check, QUERY " SELECT TIMESTAMP FROM TARGTAB"
WHERE PKCOL =?P1 ", ERROR IGNORE);
PARAMS (P1 = PKCOL)),
FILTER (CREATED_BY <> “DBA”),
FILTER (ON UPDATE, BEFORE.TIMESTAMP < CHECK.TIMESTAMP,
RAISEERROR 9999);

INSERTALLRECORDS
MAP $DATA.MASTER.SRC, TARGET $DATA.MASTER.TAREXEC,
EXCEPTIONSONLY,
COLMAP (USEDEFAULTS, ERRTYPE = "UPDATE FILTER FAILED.");
```

In the example, the query is run under the logical name of check. Values retrieved from this query can be utilized anywhere in the MAP statement by referencing check.<column name>.

The FILTER statements in the example parameter file are processed in the order that they are written. If, in the first FILTER statement, the value of the CREATED_BY column in the record being applied by Replicat is equal to the DBA account, the operation is accepted for processing by the second FILTER statement. Otherwise, it is ignored.

In this example, SQLEXEC also detects database errors, but ignores them and continues processing. This is the default behavior for ERROR.

```
SQLEXEC (ID check, QUERY " SELECT TIMESTAMP FROM TARGTAB"
WHERE PKCOL =?P1 ", ERROR IGNORE);
```

However, SQLEXEC could perform any of the following :

Syntax	Description
`ERROR REPORT`	Write the database error to a report.
`ERROR RAISE`	Enable the same error handling capabilities available for table replication errors.
`ERROR FINAL`	Enable the same error handling capabilities as ERROR RAISE, but also ignore any further queries left to process.
`ERROR FATAL`	Abend the process immediately.

In the second FILTER statement, the ON UPDATE clause directs the filter to run only for update statements. It compares the value of BEFORE.TIMESTAMP (the timestamp of the row that Replicat is attempting to apply) to CHECK.TIMESTAMP (the timestamp of the row already in the database). If the row in the database is newer than the row being applied, then the filter raises an error and the update is ignored.

In the example, the error correction was implemented by means of RAISEERROR in the SQLEXEC clause in the first MAP statement, but it could have been implemented in the second MAP statement by replacing the COLMAP clause with a SQLEXEC clause.

To handle specific issues, additional SQLEXEC statements could be executed after the filter or even between the filter statements for increased control.

Using the event log

The GoldenGate event log shows processing events, messages, errors, and warnings generated by GoldenGate. Although this information is also recorded in NonStop's Event Management System (EMS), viewing the GoldenGate log is sometimes more convenient. Use GGSCI VIEW GGSEVT command to view the event log.

Using the process report

GoldenGate generates a report about Manager, Logger, Extract, Replicat, and Syncfile at the end of each run. The report provides information about runtime parameters and statistics, including process events, and operations that were performed. The name of a process report is either MANAGER for the Manager process, or it is the same as the group name for Extract and Replicat. By default, reports are created on the subvolume GGSRPT. For example, a report for an Extract group EXTORD would have a report named GGSRPT.EXTORD.

Generate process reports with the SEND EXTRACT <group name> command and the following options:

Report	Option	Description
The end of an audit trail	AUDITEND	Queries Extract to determine whether or not all records in the TMF audit trails have been processed. This command indicates whether or not more Extract or Replicat activity must occur before a scheduled switch between databases. Until AUDITEND returns "All audit processed," more data needs to be processed before it can be assumed that secondary databases are synchronized.
Processing status	STATUS	Returns a detailed status of the processing state, including current position and activity.

Report	Option	Description
Processing statistics	REPORT	Generates an interim statistical report to the report file, including the number of inserts, updates, and deletes since the last report (default) or according to report options that can be entered. See page 82 for details on the SEND REPORT options.
TCP/IP statistics	GETTCPSTATS	Retrieves TCP/IP statistics, such as the quantity and byte length of inbound and outbound messages, the number of messages received and sent, wait times, process CPU time, and byte transmit averages.
	RESETTCPSTATS	Resets the TCP/IP statistics so the next report displays fresh statistics

Figure 32 Sample report

```
RMTTRAIL $DATA10.LOGGER.R1000038, RBA 5348453
Session Index 1
Stats started 2006/01/10 11:46:18.804165 0:00:41.522086
Local address 192.168.103.110:3319 Remote address 192.168.103.105:8004
Inbound Msgs 199 Bytes 2337, 57 bytes/second
Outbound Msgs 200 Bytes 5389492, 131451 bytes/second
Recvs 199
Sends 200
Avg bytes per recv 11, per msg 11
Avg bytes per send 26947, per msg 26947
Recv Wait Time 17592208, per msg 88403, per recv 88403
Send Wait Time 774603, per msg 3873, per send 3873
Process CPU Time 0:00:07.715372
```

NOTE In GoldenGate for NonStop, several additional reporting options are available. For specifics, see the *Reference Guide*.

Viewing process reports

To view a process report, view the file directly from the operating system's command shell, or use the VIEW REPORT command in GGSCI. You also can view process reports from the Activity Console by clicking More Info beneath the name of the process.

Storing process reports

By default, process reports are stored in the GGSRPT subvolume of the GoldenGate directory. You can designate an alternate file name by using the REPORT option of the ADD EXTRACT and ADD REPLICAT commands when you create the group from the GGSCI interface. Specify the fully qualified file name.

Once paired with Extract or Replicat, the report file may remain in its original location, or you may change its location with the ALTER command, such as:

```
ALTER EXTRACT <group name> REPORT <file name>
```

Managing process reports

Whenever a process starts, a new report file is created, and a sequence number is appended to the name of the old file. The naming sequence goes from no sequence number (current), to 0 (the most recent) to 9 (the oldest), for example: $DATA.GGSRPT.EXTORD, $DATA.GGSRPT.EXTORD0, $DATA.GGSRPT.EXTORD1 and so forth. When the file number reaches nine, the oldest file is deleted to make room for a new file, so there are never more than 11 files on the system at one time (the current report plus the ten aged reports).

To prevent the size of the report file from becoming too large, use the REPORTROLLOVER parameter in the Extract and Replicat parameter files. This parameter forces the report files to age on a regular schedule. Options are available to age the current file on a specific day and/or a specific time.

To minimize the impact of errors on the size of the Replicat report file, use the WARNRATE parameter in the Replicat parameter file. This parameter conserves the size of the report file and the event log by issuing a warning only after a specific number of errors have been generated, instead of after each one. This parameter is useful if you expect a certain number of errors and can tolerate them. The default for this parameter is to warn after 100 errors.

Generating runtime statistics

Runtime statistics show the current state of processing. By default, runtime statistics are written to the existing process report at the end of each run. To control when runtime statistics are generated, use the REPORT parameter. This parameter has options for controlling the day and time that statistics are generated.

To generate interim runtime statistics, use the SEND EXTRACT or SEND REPLICAT GGSCI command with the REPORT option syntax as shown below.

```
GGSCI> SEND <extract or replicat> <group name>
       REPORT [<time option> [RESET | FILE <name> | TABLE <name>]]
```

The <time option> controls the time span covered by the report, such as since the start of Extract or since the last report request. RESET sets the counters for that <time option> to zero. FILE or TABLE limits the report to counts for <name>. For detail on these options see SEND REPORT on page 82.

To generate runtime statistics and also cause the report file to roll over to a new one, add the ROLLREPORT option to the command, for example:

```
GGSCI> SEND EXTRACT EXTORD, REPORT
GGSCI> SEND EXTRACT EXTORD, ROLLREPORT
```

Viewing record counts

You can produce record counts at designated times during processing by using the REPORTCOUNT parameter in the Extract and Replicat parameter files. Results are printed to the process report file and to screen.

The record count shows the number of records extracted and replicated since the Extract or Replicat process started. Counts can be obtained at regular intervals or each time a specific number of records is processed.

The STATS command

To generate a statistical report for Extract or Replicat, specify the LAGSTATS parameter. GoldenGate measures lag in bytes and time:

- Lag in bytes is the difference between the position of the Extract program in the source at the time of the last checkpoint, and the current end-of-file. A lag value of UNKNOWN indicates that the process may have recently started and hasn't yet processed records, or that the source system's clock may be ahead of the target system's clock due to a reason other than time zone differences.
- Time lag reflects the lag in seconds *at the time the last checkpoint was written*. For example, if it is now 15:00:00, the last checkpoint was at 14:59:00 and the timestamp of the last record processed by the Replicat program was 14:58:00, the lag is reported as 00:01:00 (one minute, the difference between 14:58 and 14:59).

The report includes the following general performance categories:

- General statistics.
- Lag statistics.
- Extract's processing in the GoldenGate trail.
- Audit trail reading statistics for Extract (when applicable).
- Output statistics for Extract only.

The following table describes each item in the lag statistics report.

Item	**Description**
Last Record Timestamp	The timestamp of the source record (when the source record was input or committed to the target database).
Configured Interval	Determined by the LAGSTATS INTERVAL parameter.
Actual Duration	The duration of time measured.
Records Processed	Number of records output or replicated during the period.
Records per Second	Records processed per second during the interval.
Source Records per Second	The estimated rate of records read for either the TMF audit trails or logger processes.
Last lag	The time lag of the last record measured between (1) the records update into the source database and (2) the actual processing of the record by Extract or Replicat.
Min lag	Smallest value of Last lag during the interval.

Item	**Description**
Average lag	Average time lag during the interval for all records processed.
Peak lag	Peak time lag during the interval for all records processed and the timestamp of the peak.
Last est. record lag	An estimate of the number of records the component is behind the current record.
Pct Below Lag of mi:ss:mmm	The percentage of times lag was below the time threshold you specified. This is an optional statistic, which can occur up to five times. To generate the statistic, specify the THRESHOLD option for LAGSTATS.
Pct CPU Busy	The amount of time the CPU in which Extract or Replicat was running was busy during the interval.
PCT Process Busy	The amount of time Extract or Replicat was busy during the interval.
At EOF?	Whether or not more data was available to process the last time more data was requested by Extract or Replicat from the audit or GoldenGate trails.
Trail Reads per Second	When reading GoldenGate trails, the number of attempted block reads per second.
Bytes per Trail Read	When reading GoldenGate trails, the number of bytes read per successful read.
Records per Block Read	When reading GoldenGate trails, the number of records read per successful read. This indicates the blocking factor on input.
Wait per Block Read	When reading GoldenGate trails, the amount of time Extract or Replicat waits, on average, to complete the read.
Audit Bytes per Second	The number of bytes of audit processed per second (TMF Extract only).
Pct EOF Trail Reads	For TMF Extract, the percentage of times Extract reached the end of file, compared with the number of records processed. For Replicat or Extract reading GoldenGate trails, the number of times the process read at the end of file, compared with the total number of blocks it attempted to read.
Transactions per Second	For TMF Extract, the number of transactions processed per second.
Transactions Aborted	For TMF Extract, the number of transactions aborted during the interval.

Item	**Description**
Audit positions	The number of times during the interval that Extract requested Audserv to position for read.
Audit position seconds	The elapsed time in seconds required for Audserv to position for read.
Audserv requests	The number of data requests to Audserv during the interval.
Audserv request wait seconds	The elapsed time for the Audserv to fulfill data requests during the interval.
Long transactions	The number of long transactions during the interval.
Long transaction seconds	The elapsed time for the long transactions that occured during the interval.
Output Bytes per Second	For Extract, the bytes of data output to the extract trails per second.
Output Blocks per Second	For Extract, the number of blocks of data written to the GoldenGate trails per second.
Records per Block Written	For Extract, the average number of records in each block written.
Bytes per Block Written	For Extract, the average number of bytes in each block written.
Wait per Block Written	For Extract, the amount of time waiting for the last write or TCP/IP send to complete before sending the next block of data. This statistic can provide an idea of whether or not the network might be introducing a delay to Extract.
Average Record Flush Delay	For Extract, the estimated average amount of time a record was held in its buffers before flushing.
Pct Output/Input	For Extract, the ratio of bytes output compared with input bytes processed. Useful for estimating potential bandwidth required by Extract based on amount of TMF audit generated.

Collecting events from other systems

Event messages created by the Collector and Replicat on Windows and UNIX systems can be captured and sent back to EMS on NonStop systems. This feature enables centralized viewing of GoldenGate messages across platforms.

To collect events from other systems:

1. Run Collector on NonStop to collect and distribute EMS messages. For each EMSCLNT process, run one Collector process. The following example runs Collector and outputs its messages to $0.

   ```
   TACL> ASSIGN STDERR, $0
   TACL> RUN SERVER /NOWAIT/ -p 7880
   ```

2. Run the EMSCLNT utility on the remote target. EMSCLNT reads a designated error log and runs indefinitely, waiting for more messages to send. When EMSCLNT receives a message, it sends the message to a TCP/IP collector process on NonStop. See the examples for running EMSCLNT on other operating systems for syntax information.

Running EMSCLNT on other operating systems

This Unix example reads the file ggslog.err for error messages. Error messages are sent to the collector to the NonStop at IP address 13.232.123.89 listening on port 7850. The Collector on NonStop writes formatted messages to EMS Collector $0.

Example

```
> $emsclnt -h 13.232.123.89 -p 7850 -f ggserr.log -c $0
```

This Windows example (from the DOS prompt) reads the file d:\ggserrs\log.txt for error messages. Error messages are sent to the Collector on host ggs2 listening on port 9876. The Collector on NonStop writes formatted messages to EMS Collector $P0.

Example

```
> emsclnt -h ggs2 -p 9876 -f c:\ggs\ggserr.log -c $P0
```

Argument	Description
`-h ggs2`	The node on which the collector is being run. Can be a name or IP address. This is a required parameter.
`-p 9876`	The port at which the collector is listening for messages. This is a required parameter.
`-f c:\ggs\ggserr.log`	The error file from which EMSCLNT retrieves error messages. This is a required parameter.
`-c $P0`	The collector to which EMS messages should be written on the NonStop (default is $0).

CHAPTER 11

Using GoldenGate Utilities

GoldenGate supplies utilities that support a variety of processing requirements, such as generating data definitions and DDL, the ability to back records out of a target, and a streamlined method for duplicating files when incremental change processing is not necessary.

Generating data definitions with DEFGEN

When capturing, transforming, and delivering data across disparate systems and databases, you must understand both the source and target layouts. Understanding column names and data types is instrumental to GoldenGate's data synchronization functions.

The DEFGEN utility produces a file defining the source files and tables' layouts. These definitions are used by the Collector and by Replicat. In some cases, Extract also uses a definition file containing the target layouts when transformation operations are required on the source system.

The output definitions are written and saved to a text file and transferred to all target systems in text format. When they start, Replicat and the Collector read the definitions to interpret the data formats read from GoldenGate trails.

NOTE Do not modify the text file that is output from DEFGEN.

Once you have generated your definitions, you must specify their location in your process' parameter file. Replicat uses the SOURCEDEFS parameter to indicate which source definition file to use. Collector uses the -d argument at startup to specify which source definition file to use.

Run DEFGEN interactively or using a batch obey script.

Syntax

```
TACL> RUN DEFGEN [/IN <command file>/] [EXCLUDESYSTEM] [EXPANDDDL <options>]
[RECORDNAMEPROMPTING]
```

Option	Description
`IN <command file>`	If you have created and saved a parameter file using the NonStop editor, enter the name of that file.
`EXCLUDESYSTEM`	Causes DEFGEN to omit the NonStop system name from the files and tables for which definitions are being generated.

Option	Description
`EXPANDDDL <options>`	Use the EXPANDDDL parameter to manipulate output for Enscribe record definitions containing arrays and redundant field names. This feature is primarily useful when mapping Enscribe files to SQL tables. It can also be useful when generating SQL tables based on Enscribe definitions using the DDLGEN utility. EXPANDDDL is not necessary when the source database is NonStop SQL.
`RECORDNAMEPROMPTING`	RECORDNAMEPROMPTING allows you to enter the name of an existing record definition to use when generating a definition for a new table. Use this parameter to point to the same definition for multiple tables with identical definitions made up of the same columns, column order, and data types.

For details about the DEFGEN parameters, see the *Reference Guide*.

Configuring DEFGEN interactively

1. Run DEFGEN from TACL using the following syntax:

```
TACL> RUN DEFGEN EXPANDDDL EXPANDGROUPARRAYS RESOLVEDUPGROUP OMITREDEFS
```

2. In response to the prompts, enter information similar to the following example:

For the prompt:	Enter:
Enter definitions file name (or Exit):	`$DATA1.GGSDEF.CUSTDEF`
File/Table to create definition for (or Exit):	`$DATA1.GGSSOU.ECUSTMER`
Include DDL record definition (Y/N)?	`Y`
DDL dictionary:	`$DATA1.GGSDDL`
DDL record definition name:	`ECUSTMER-REC`
File/Table to create definition for (or Exit)	`EXIT`

3. Transfer this file, as a text file, to the target system.

Configuring DEFGEN in batch

1. Use the NonStop editor to create a parameter file.
2. Enter parameters similar to the following examples:
   - For NonStop SQL

```
$DATA1.GGSDEF.CUSTDEF
$DATA1.GGSSOU.TCUSTMER
EXIT
```

- For NonStop Enscribe

```
$DATA1.GGSDEF.CUSTDEF
$DATA1.GGSSOU.ECUSTMER
Y
$DATA1.GGSDDL
ECUSTMER-DEF
EXIT
EXIT
```

3. Initiate DEFGEN from TACL using a syntax similar to:

```
TACL> RUN DEFGEN /IN GGSPARM.DEFGEN/
```

4. Transfer the generated definitions file, as a text file, to the target system.

A sample definitions file

```
Definition for table $DATA1.GGSSOU.TCUSTMER
Record length: 198
Syskey: 0
Columns: 13
TS     134 8 0 0 0 0 1 8 8 8 0 0 0 0 1 0 1
RECNUM 132 4 8 0 0 0 1 4 4 4 0 0 0 0 1 0 1
SYSNAME 1 8 12 0 0 0 0 8 8 8 0 0 0 0 1 0 0
TEXT   0 64 20 0 0 0 0 64 64 64 0 0 0 0 1 0 0
VAL1   134 8 84 0 0 0 1 8 8 8 0 0 0 0 1 0 0
VAL2   134 8 92 0 0 0 1 8 8 8 0 0 0 0 1 0 0
COL_COMPUTE 134 8 100 0 0 0 1 8 8 8 0 0 0 0 1 0 0
I16    130 2 108 0 0 0 1 2 2 2 0 0 0 0 1 0 0
I32    132 4 110 0 0 0 1 4 4 4 0 0 0 0 1 0 0
I64    134 8 114 0 0 0 1 8 8 8 0 0 0 0 1 0 0
I32_TOTAL 132 4 122 0 0 0 1 4 4 4 0 0 0 0 1 0 0
JTS    134 8 126 0 0 0 1 8 8 8 0 0 0 0 1 0 0
JTS_TEXT 0 64 134 0 0 0 0 64 64 64 0 0 0 0 1 0 0
End of definition
```

Running DEFGEN to use existing definitions

Multiple tables that have exactly the same structure (identical columns, column order, and data types), can use the same definition. To run DEFGEN for these tables, use the RECORDNAMEPROMPTING argument.

1. Run DEFGEN from TACL using the following syntax:

```
TACL> RUN DEFGEN RECORDNAMEPROMPTING
```

2. In response to the prompts, enter information similar to the following example:

For the prompt:	Enter:
Enter definitions file name (or Exit):	`$DATA1.GGSDEF.CUSTDEF`
File/Table to create definition for (or Exit):	`$DATA1.GGSSOU.ECUSTMER`

For the prompt:	**Enter:**
Use record name for definition file (Y/N)?	`Y`
Record or definition name to be used:	`CUSTOMER-DEF`
File/Table to create definition for (or Exit)	`EXIT`

Creating target database DDL

DDLGEN generates table definitions for target databases based on existing Enscribe and NonStop SQL definitions. It can also use the output from the DEFGEN utility (see above). DDLGEN reduces the work necessary to create databases on platforms such as UNIX or Windows, and enables the creation of NonStop SQL databases based on Enscribe definitions. Target templates are provided for NonStop SQL, Oracle, SQL Server, DB2, and Sybase.

DDLGEN is initiated either interactively by supplying responses to user prompts, or in batch mode by supplying an input file.When running interactively, the user is supplied several prompts. For batch execution, the answers to the prompts are supplied in the obey file. It is recommended that the user runs the process in interactive mode to better understand the replies to specify in the obey file.

The result of running DDLGEN is a text file containing the create table statements. Transfer this file, as a text file, to your target system.

For details about the DDLGEN parameters, see the *Reference Guide*.

The DDLGEN syntax is:

```
TACL> RUN DDLGEN [/IN <command file>/] [-d <DEFGEN output>]
```

Argument	**Description**
`IN <command file>`	If you have created and saved a file of responses using the NonStop editor, enter the name of that file. See the information on configuring DDLGEN interactively on page 143 for a list of the responses.
`-d <DEFGEN output>`	Instructs DDLGEN to use the definitions file produced by DEFGEN.

Configuring DDLGEN interactively

1. Initiate DDLGEN from TACL using a syntax similar to:

   ```
   TACL> RUN DDLGEN
   ```

2. In response to the prompts, enter information similar to the following example:

For the prompt:	**Enter:**
Output file for table DDL (or Exit):	`$DATA1.GGSDEF.CUSTDEF`

For the prompt:	Enter:
DDL template file name (or Exit):	`<template name>` There are seven different templates prepackaged with GoldenGate: TMPLDB2 - DB2 for Windows and UNIX TMPLDB2M - DB2 for the z/OS and OS/390 TMPLMSA - MS Access TMPLMSS - MS SQL Server TMPLORA - Oracle TMPLSYB - Sybase TMPLTDM - NS SQL/MP
Source File/Table (or Exit):	`$DATA1.GGSSOU.ECUSTMER`
DDL dictionary:	`$DATA1.GGSDDL`
DDL record definition name:	`ECUSTMER-REC`
Source File/Table (or Exit):	`$DATA1.GGSSOU.TCUSTORD`
Source File/Table (or Exit):	`EXIT`

3. Transfer the definitions file, as a text file, to the target system.

Configuring DDLGEN in batch

1. Use the NonStop editor to create a parameter file. For this example, the file name is GGSPARM.DDLGEN.
2. Enter parameters similar to the following examples:

```
$DATA1.GGSDEF.CUSTDEF
TMPLORA
$DATA1.GGSSOU.ECUSTMER
$DATA1.GGSDDL
ECUSTMER-REC
$DATA1.GGSSOU.TCUSTORD
EXIT
```

3. Initiate DDLGEN:

```
TACL> RUN DDLGEN /IN GGSPARM.DDLGEN/ -d $DATA1.GGSDEF.CUSTDEF
```

   In this step, you are instructing DDLGEN to use the DEFGEN definitions file that was produced by a previous run of DEFGEN.
4. Transfer the generated definitions file, as a text file, to the target system.

Addressing Enscribe DDL peculiarities

Enscribe record definitions often contain the following items that do not map directly to SQL environments:

- OCCURS items.

  SQL columns cannot have multiple occurrences, while Enscribe fields can.
- Group level items.

  There is no grouping hierarchy in SQL, while an Enscribe record definition can contain fields that are redundant until qualified at the group level. This means that redundant column names can be created when mapping Enscribe definitions to SQL. For example, an Enscribe record might contain the field YEAR twice, once within the BEGIN-DATE group and once within the END-DATE group.

To get around these conditions, you can run the DEFGEN utility with various EXPANDDDL options set, as described in the *Reference Guide*. Use the -d parameter to specify the definitions file created by DEFGEN as input to DDLGEN.

Understanding the template file

Templates are provided with each version of GoldenGate. A template file specifies how to generate the target definitions based on the source definitions. Each template file contains the following items:

- Literal text to output for each table definition.
- Source to target data type conversion specifications.
- Column name substitution specifications.
- Miscellaneous run-time parameters.
- Section headers.
- Comments, which begin with a pound sign (#).
- *Session parameters*, which are resolved at run-time by user prompts and applied during the entire DDLGEN session. Session parameters begin with a question mark (?).
- *Per-table parameters*, which are input by the user for each table definition generated. Per-table parameters begin with a percent sign (%).
- *Calculated parameters*, which include information determined by DDLGEN. Calculated parameters include the following:

Calculated Parameter	Description
`?TABLE`	The file name portion of the source table or file.
`?COLUMNS`	A list containing each column, its target data type, precision and scale (if any), and null/not null syntax.
`?KEYCOLUMNS`	A list containing each column in the primary key.

`?MAXPAGES`
`?MAXMEGS`
`?CURPAGES`
`?CURMEGS`
	The maximum and current number of 2048-byte pages and megabytes in the source table.

A sample template file

This sample (TMPLORA), is a template file for converting NonStop DDL to Oracle DDL.

The sections are:

- Table creation section specifying operations for creating and managing tables. Note that in this section ?TABLE, ?COLUMNS and ?KEYCOLUMNS are resolved by DDLGEN. ?TABLE_SPACE is prompted for once and will apply to every table, while %NEXT_SIZE is prompted for on a per-table basis.
- Column name mapping section containing source NonStop and target Oracle column names. This section maps the source column names to the target names. In this example, any occurrences of ROWID in the NonStop database will be changed to ROWID_ in the Oracle definition. If Oracle keywords appear in your NonStop database definitions, add entries to this list.
- Miscellaneous parameters section. Specify instructions for column formatting.
- Column type mapping section. Determines how NonStop types are defined in Oracle. Precision and Scale definitions can be YES, NO or a constant, positive value.

Note in this example, there are two entries for both CHAR and VARCHAR. Because Oracle allows a maximum of 255 characters in a VARCHAR2, we specify that all instances of CHAR and VARCHAR with length greater than 255 should become LONGS.

Figure 33 Sample template file for creating Oracle table DDL

```
# Table Creation Section
#
DROP TABLE ?TABLE;
CREATE TABLE ?TABLE
(
?COLUMNS
,CONSTRAINT PK_?TABLE
PRIMARY KEY
(
?KEYCOLUMNS
)
USING INDEX
TABLESPACE ?TABLE_SPACE
)
TABLESPACE ?TABLE_SPACE
STORAGE (INITIAL 50K NEXT %NEXT_SIZE);
#
# Column Name Mapping Section
#
# NonStop column name     Oracle target name
#
ROWID           ROWID_
SYSDATE         SYSDATE_
#
# Miscellaneous Parameters Section
#
INCLUDENULL
#
# Column Type Mapping Section
#                                         Max    Max
# NonStop TypeTarg DB TypePrecision  Scale  Prec.  Scale
#
CHAR      VARCHAR2      Y            N      255    N
CHAR      LONG          Y            N      N      N
VARCHAR   VARCHAR2      Y            N      255    N
VARCHAR   LONG          N            N      N      N
REAL      NUMBER        N            N      N      N
DOUBLE    NUMBER        N            N      N      N
NUMERIC   NUMBER        Y            Y      N      N
MALLINT   NUMBER        Y            N      N      N
INTEGER   NUMBER        Y            N      N      N
LARGEINT  NUMBER        Y            N      N      N
DECIMAL   NUMBER        Y            Y      N      N
DATE      DATE          N            N      N      N
TIME      DATE          N            N      N      N
TIMESTAMP DATE          N            N      N      N
DATETIME  VARCHAR2      Y            N      N      N
INTERVAL  VARCHAR2      Y            N      N      N
```

Sample NonStop SQL table definition.

```
CREATE TABLE TCUSTORD
(
    CUST_CODE        CHAR (4)                    NOT NULL,
    ORDER_DATE       DATETIME YEAR TO SECOND     NOT NULL,
    PRODUCT_CODE     CHAR (8)                    NOT NULL,
    ORDER_ID         NUMERIC (18)                NOT NULL,
    PRODUCT_PRICE    DECIMAL (8,2),
    PRODUCT_AMOUNT   DECIMAL (6,0),
    TRANSACTION_ID   NUMERIC (18)                NOT NULL,
    DESCRIPTION      CHAR (400),
    PRIMARY KEY (CUST_CODE, ORDER_DATE, PRODUCT_CODE, ORDER_ID)
);
```

Modifying the template file

In this example, you can make three modifications:

1. Add a DATE_MODIFIED column to each table, as well as to the primary key.
2. Calculate the NEXT value based on the table's current size. Note that any parameter value that evaluates to a numeric value can be multiplied or divided.
3. Substitute the column name ORDER_NUM for instances of ORDER_ID.

The following is the template file, with modifications shown in bold.

```
#
# Table Creation Section
#
DROP TABLE ?TABLE;
CREATE TABLE ?TABLE
(
?COLUMNS
,  DATE_MODIFIED DATE         NOT NULL
,  CONSTRAINT PK_?TABLE
   PRIMARY KEY
   (
?KEYCOLUMNS
,  DATE_MODIFIED
   )
   USING INDEX
   TABLESPACE ?TABLE_SPACE
)
TABLESPACE ?TABLE_SPACE
STORAGE (INITIAL 50K NEXT ?CURPAGES/5K);
#
# Column Name Mapping Section
#
# NonStop column name     Oracle target name
#
ROWID                  ROWID_
SYSDATE                SYSDATE_
ORDER_ID               ORDER_NUM
```

Example session to generate Oracle definition.

```
1> RUN DDLGEN
Output file for table DDL (or Exit): ORADDL
DDL template file name (or Exit): TMPLORA
Value for param TABLE_SPACE: USERS
Source File/Table (or Exit): $DATA1.SAMPLE.TCUSTORD
Source File/Table (or Exit): EXIT
```

Contents of ORADDL after DDLGEN is run.

```
DROP TABLE TCUSTORD;
CREATE TABLE TCUSTORD
(
   CUST_CODE           VARCHAR2            (4)    NOT NULL
,  ORDER_DATE          DATE                       NOT NULL
,  PRODUCT_CODE        VARCHAR2            (8)    NOT NULL
,  ORDER_NUM           NUMBER              (18)   NOT NULL
,  PRODUCT_PRICE       NUMBER              (8,2)  NULL
,  PRODUCT_AMOUNT      NUMBER              (6,0)  NULL
,  TRANSACTION_ID      NUMBER              (18)   NULL
,  DATE_MODIFIED       DATE                       NOT NULL

,  CONSTRAINT PK_TCUSTORD
   PRIMARY KEY
   (
   CUST_CODE
,  ORDER_DATE
,  PRODUCT_CODE
,  ORDER_NUM
,  DATE_MODIFIED
   )
   USING INDEX
   TABLESPACE USERS
)
TABLESPACE USERS
STORAGE (INITIAL 50K NEXT 470K);
```

GoldenGate rollback

You can run GoldenGate for selective rollback processing when synchronizing data from TMF applications. GoldenGate rollback uses before images from the TMF audit trail to undo database changes for user-defined tables and files, rows and records, and time periods. Two valuable applications of rollback are:

- Reversing errors caused by corrupt data or accidental deletions. For example, if you issue a SQL UPDATE or DELETE command without the WHERE clause, rollback reverses the errors caused by the erroneous operations.
- Maintaining large test databases. Use rollback to restore the test database to its state before a test run, enabling test cycles to run quickly. Since the rollback process only reverses changes, a database can be restored much more efficiently than a complete database restore.

The Reverse program prepares extracted data for rollback by:

- Reversing the ordering of database operations in the file so Replicat processes them in reverse order, guaranteeing that records with the same key are properly applied.
- Changing the before or after image indicator in each record.
- Changing delete operations to inserts, and insert operations to deletes.
- Reversing the BEGIN and END transaction indicators to delimit each transaction.

Syntax `RUN REVERSE <source extract trail> <target extract trail>`

Argument	Description
`<source extract trail>`	The EXTTRAIL specified in the Extract parameter file. The REVERSE program can also run against a set of trails.
`<target extract trail>`	The EXTTRAIL specified in the Replicat parameter file.

NOTE <source extract trail> and <target extract trail> must be different.

Using GoldenGate rollback

1. Create an Extract parameter file.

Sample Extract parameter file

```
SPECIALRUN
BEGIN 1996-05-30 17:00
END   1996-05-30 18:00
GETUPDATEBEFORES
EXTFILE $DATA2.DAT.EXTSRC
TABLE $DATA3.TABLES.SALES;
TABLE $DATA3.TABLES.ACCOUNTS;
```

About the parameters:

- SPECIALRUN indicates that no checkpoints are required.
- BEGIN and END parameters specify the period of time during which the original transactions occurred.
- GETUPDATEBEFORES directs Extract to extract before images.
- EXTFILE specifies the trail holding the extracted images is an Extract flat file.
- FILE and TABLE parameters specify the tables and files to reverse. Include special criteria, such as ACCOUNT = "CA".

2. Create a Replicat parameter file, similar to example on page 150.

Sample Replicat parameter file

```
SPECIALRUN
END RUNTIME
EXTFILE $DATA2.DAT.EXTTARG
MAP $DATA3.TABLES.SALES, TARGET $DATA4.TABLES.SALES;
MAP $DATA3.TABLES.ACCOUNTS, TARGET $DATA4.TABLES.ACCOUNTS;
```

About the parameters:

- SPECIALRUN indicates that no checkpoints are required.
- EXTFILE identifies the Extract trail as a flat file.

- MAP specifies mappings from source to target. When backing out the original database, the source and target are the same. When backing delivered transactions out of the target database, the source and target of each map are different.

3. Run Extract, Reverse, and Replicat similar to:

```
TACL> RUN EXTRACT /IN $DATA1.MYSUB.EXTPARM/
TACL> RUN REVERSE $DATA2.DAT.EXTSRC $DATA2.DAT.EXTTARG
TACL> RUN REPLICAT /IN $DATA1.MYSUB.REPPARM/
```

   - Running Extract extracts the changes.
   - Running Reverse prepares the extracted changes for replication.
   - Running Replicat applies the before images to the original database (or to an alternative database if you are reversing delivered changes).

Reviewing reversed records

Before applying Rollback, you may want to review your pending changes. To review, run Extract with the FORMATSQL parameter. The Extract file contains SQL syntax describing each of the statements to be reversed.

Correcting unexpected results

If rollback produces unexpected or undesired results, you can reapply the original changes to the database. Use the source Extract file (<source extract file>) specified in the REVERSE command as the EXTFILE entry in your Replicat parameter file and run Replicat again.

Invalid conditions for rollback

Rollback does not work in the following circumstances:

- Because entry-sequenced does not support deletes, rollback does not work when inserts are reversed for an entry-sequenced table or file.
- When a table is organized with a cluster key (a non-unique primary key), reversing deletes results in SYSKEY values that are different from the original rows. The delete's before image is applied as an insert into the table. Replicat cannot control the SYSKEY value, so it becomes the timestamp when the records are reversed and relationships depending on the original SYSKEY are lost. In addition, if both deletes and inserts are reversed, results will likely be incorrect. If a set of columns can be defined that uniquely identifies the record, you can use the KEYCOLS option of the Replicat MAP statement.

Using Syncfile

Syncfile manages non-database file duplication. For example, you may want to replicate configuration files, which are small and change infrequently. This is a common requirement for maintaining a secondary system that may see frequent database changes, but infrequent configuration file changes.

Syncfile can copy almost any type of file, making it suitable for other scenarios which require only infrequent, off-hours copying. By default, Syncfile uses the NonStop FUP DUP utility to perform file duplication; however, it can also invoke user-written TACL scripts to perform more specialized file duplication, such as FTP over TCP/IP.

Syncfile is easy to implement; simply define its set of parameters. The two main parameters include a file list to duplicate, and one to many schedules. A file set can be a file name, a wildcarded file name, or a file exclude list. The schedules are events which can be as frequent as you wish, such as every day, every hour, or every ten minutes.

Syncfile options control the following:

- The files to duplicate.
- The schedule for determining when files should be duplicated.
- The method for duplication (FUP, FTP, etc.).
- Whether files should be always duplicated, or only when modified.

A Syncfile parameter file can contain multiple schedules and file sets. In addition, you can create multiple Syncfile processes to support duplication for different applications or other requirements.

Syncfile processes are persistent. If a Syncfile process goes down unexpectedly, Manager automatically restarts it.

Implementing Syncfile

1. Create a parameter file with the necessary Syncfile parameters. The parameter file includes the names of the files to duplicate, the schedules, and other options.

A sample Syncfile parameter file

```
EVENT DAILY, EVERY DAY AT 1:00, EXCLUDE FRIDAY, EXCLUDE AUGUST 2;
EVENT FREQUENT, EVERY 2 HOURS;

DUP $DATA1.SOURCE.*, TARGET \BKUP.$DATA2.*.*,
ALWAYS, EVENT DAILY;

DUP $DATA2.GGSPARM.*, TARGET \BKUP.$DATA3.*.*,
TACLCMD "RUN $DATA1.GGSTACL.SYNCTCL <source> <target>",
CHANGED, EVENT FREQUENT;
```

This parameter file specifies the following attributes and actions:

- Two events—daily and frequent. The daily event happens every day at 1:00 AM. However, the daily event is cancelled on all Fridays and August 2. The frequent event occurs every two hours. There are two DUP specifications. The first DUP specification indicates the file set $data1.source.*. Files satisfying that description are duplicated according to the daily event schedule (every day at 1:00). These files are duplicated regardless of whether or not the data has changed (the “always” option). Files are duplicated to \bkup.$data2 with the same subvolume and file name as the corresponding source files.
- By default, the FUP DUP <source>, <target>, PURGE, SAVEALL command is used to duplicate the files.
- The second DUP specification names everything from $data2.ggsparm to be copied to \bkup.$data3 with the same subvolume and file names. Files are duplicated on the frequent event schedule (every two hours). However, only those files with a

modification timestamp on the source greater than that of the target will be duplicated (the changed option). The changed option does not have to be specified since it is the default option.

In this example TACLCMD is added as a clause to the DUP parameter. This will cause Syncfile to invoke the TACL macro $data1.ggstacl.synctcl to duplicate the file. The macro is responsible for determining how to move source files to the target system, as well as any intermediate required steps.

In this instance, the <source> and <target> arguments should not be replaced with any file name. They act as keywords to trigger Syncfile to use the DUP $DATA2.GGSPARM.*, TARGET \BKUP.$DATA3.*.* statement to identify the source and target parameters that will be passed to the macro.

If file names are entered in <source> and <target> they will be passed to the macro instead. The following TACL macro and Syncfile parameters will pass GGSPARM.FILE1 as %1% and GGSPARM.FILEB as %2% to cause it to duplicate FILE1 to FILEB, not to $DATA3.GGSPARM.FILEA.

```
?TACLMACRO
FUP DUP %1%, %2%

DUP $DATA1.GGSPARM.FILE1, TARGET $DATA3.GGSPARM.FILEA,
TACLCMD “RUN $DATA1.GGSPARM.TACL1 GGSPARM.FILE1 GGSPARM.FILEB”,
ALWAYS, EVENT DAILY 1330;
```

NOTE Leaving out the <SOURCE> and <TARGET> arguments will cause Syncfile to abend.

- The two most important parameters for Syncfile are EVENT and DUP. At least one EVENT and one DUP parameter are required for each Syncfile operation. Each parameter entry must be terminated with a semi-colon (;).

2. Start GGSCI and add the Syncfile process.

```
TACL> RUN GGSCI
GGSCI> ADD SYNCFILE <group name>
[, PARAMS <parameter file>]
[, REPORT <report file name>]
[, PROCESS <process name>]
[, PROGRAM <program name>]
```

Add options as desired.

3. Start the Syncfile process.

```
GGSCI> START SYNCFILE
```

See the *Reference Guide* for details about the GGSCI Syncfile commands and the Syncfile parameters.

CHAPTER 12
Collecting and Reporting Statistics

GoldenGate provides MEASFLS and MEASRPT to assist in determining sizing and configuration parameters for TMF and non-TMF based data synchronization. MEASFLS gathers statistics using Measure. MEASRPT interprets Measure statistics and produces a report that assists in the sizing process.

Running MEASFLS

The function of MEASFLS is to start Measure processes that collect data about file I/O at specified intervals.

MEASFLS uses a parameter file to identify the list of files to measure, the reporting period, and other parameters. Create or edit the parameter file using the NonStop editor.

Run MEASFLS using the following syntax.

```
TACL> RUN MEASFLS /IN <param file>/
```

MEASFLS initiates activity measurements. After MEASFLS finishes, wait for Measure activity to complete before running MEASRPT.

Example MEASFLS parameter file

```
--
-- MEASFLS Parameters
--
START 2006-03-07 01:00
DURATION 7 DAYS
INTERVAL 1 HOUR
MEASFILES $DATA2.MEASDAT.MDAT
PURGEMEASFILES ON
WILDCARD $*.*.*
```

This example parameter file:

- Runs Measure for seven days with a reporting interval of one hour, starting at March 7, 2006 at 01:00.
- Gathers statistics for all files on the system every hour.
- Extracts Measure data into files named $DATA2.MEASDAT.MDATnn.
- Purges the previous measurement using the same Measure data files.

MEASRPT statistical output

The MEASRPT report outputs the statistics generated with MEASFLS and Measure. These statistics are output during individual intervals as cumulative totals, identifying both peak and summary activity levels.

Many of the statistics can be displayed as either absolute numbers or on a per-second basis. See the MEASRPT parameter REPORTRATE in the *Reference Guide*.

Statistic Category	Description
CPU Busy	Shows the average percent busy across processors during each period and over the total period measured. The report includes a peak utilization rate. These figures provide insight into the current utilization rates on the system and can be used to project utilization rates after GoldenGate is installed and running.
Top 10 Peak Periods	This section of the report displays the top 10 most active periods on the system as a whole, according to three criteria. 1. The total number of output operations across all files (inserts, updates and deletes). 2. The number of bytes generated during the period, without compression, plus 48 bytes per record. This number approximates the number of bytes that would be transferred in a given period to the target system if compression is not used. Each record transferred by GoldenGate is preceded by a 48-byte header. 3. The number of bytes generated during the period, accounting for compression on update records, plus 48 bytes. This number is an estimate of the number of bytes that would be transferred to the target system over the period. These numbers, especially the last two, provide an estimate of the communications bandwidth requirements.
Top 10 Peak Files	This section of the report displays the most active files over a single interval. For non-TMF applications, distributing files across multiple Logger processes becomes a matter of figuring out the total number of I/Os a single Logger can handle, then assigning files to each Logger accordingly. This section of the report helps identify how files might be assigned to multiple trails and audit trails, and downstream, over different communications channels.
Statistics Across All Files	This section of the report displays a summary of activity across all files for each interval of the report. These statistics point out peak I/O activity, and aid in sizing bandwidth requirements. This section also reports the peak byte generation over the intended retention period. Use this to size total disk space requirements in the event of an extended outage on the target system. You can alter the intended retention period with the MEASRPT RETENTION parameter.

Statistic Category	Description
Statistics By File	This section of the report displays statistics for each interval for every file extracted and selected for the report. You can exclude specific files from the report (with the EXCLUDEFILES parameter), even after the statistics have been gathered. Excluding files can be useful if not all data will be delivered. Statistics for each file include cache hit rate, file busy figures, bytes generated, and the number of operations. Generally, Top 10 Peak Files provides enough information for distributing the load effectively, but occasionally more detail is desired. You can limit statistics by file to the most active files (LISTLIMIT parameter) or omit them completely (FILEDETAILS parameter).
Statistics by Program	This section of the report displays statistics for each interval for every program that updates files specified in the report (if a program updates files excluded from the report, those statistics are not included in the report either). Understanding which programs are causing the most updates can be useful for spotting opportunities to eliminate certain delivery activity. For example, a FUP LOAD on one system may not be required on the other system. You can exclude specific programs from the report (using the EXCLUDEPROGRAMS parameter). When a program is excluded from the report, file activity caused by the program is excluded from all report sections. You can limit statistics by program to the most active programs (LISTLIMIT parameter) or omit them completely (PROGDETAILS parameter).
TMF Transaction Report	This section of the report displays statistics regarding TMF transactions during each interval and in total. This section also displays the programs that began the most transactions and programs that aborted the most transactions.

Running MEASRPT

MEASRPT outputs statistics gathered by MEASFLS that are pertinent to the GoldenGate sizing process.

MEASRPT uses a parameter file to identify the list of files to report about, activity reporting criteria, the reporting period, and other parameters. Create or edit the parameter file using the NonStop editor.

Run MEASRPT from TACL using the following syntax.

```
TACL> RUN MEASRPT /IN <param file>, OUT <report file>/
```

Example MEASRPT parameter file

```
--
-- MEASRPT Parameters
--
MEASFILES $DATA2.MEASDAT.MDAT
INCLUDEFILE $*.GGSDAT.*
GETSQL OFF
GETAUDITED OFF
LISTLIMIT 30
```

The example parameter file results in the following:

- Specifies that Measure data extracted into files named $DATA2.MEASDAT.MDATnn be used for the report.
- Reports file activity for files in subvolume GGSDAT on any volume, including secondary partitions (even though more data may have been extracted).
- Ignores (by default) alternate key files, SQL indexes, and NonStop files (codes 1-1000).
- Explicitly omits SQL tables from the report.
- Omits audited files from the report.
- Limits detailed output to the 30 most active files and programs.

CHAPTER 13

Using the Logdump Utility

This chapter contains instructions for using Logdump, a utility that enables you to search for, filter, view, and save data that is stored in a GoldenGate trail or extract file. Logdump is used for troubleshooting and generally should be used with guidance from a GoldenGate support analyst.

The first part of the chapter provides instructions for using the utility. The second part of the chapter contains a reference guide to all Logdump commands in alphabetical order.

Getting started with Logdump

This section introduces you to basic Logdump commands that enable you to open files, control the display, navigate through a file, and filter for specific information, among other basic tasks. It also illustrates and explains the components of a record.

Following this section is an alphabetical reference of the Logdump commands.

Viewing the first record

These steps show you how to set up the Logdump environment and start viewing records.

To run Logdump

Run the logdump program from the GoldenGate installation location. Logdump command lines are numbered so that you can use edit and history commands.

To set up the view

The following commands set up a Logdump environment that shows the information most commonly used when analyzing GoldenGate trail records.

1. To view the record header with the data:

   ```
   Logdump 1> GHDR ON
   ```

   The record header contains information about the transaction.

2. To add column information:

   ```
   Logdump 2> DETAIL ON
   ```

   Column information includes the number and length in hex and ASCII.

3. To add hex and ASCII data values to the column information:

   ```
   Logdump 3> DETAIL DATA
   ```

4. To view user tokens:

   ```
   Logdump 4> USERTOKEN ON
   ```

   User tokens are custom user-defined information that is specified in a TABLE or FILE mapping statement and stored in the trail file for specific purposes.

5. To control how much record data is displayed:

   ```
   Logdump 5> RECLEN <length>
   ```

To open a trail file

1. Open a file with the following command:

   ```
   Logdump 6> OPEN <file_name>
   ```

   Where: <file_name> is either the relative name or fully qualified name of the file, including the file sequence number. For example:

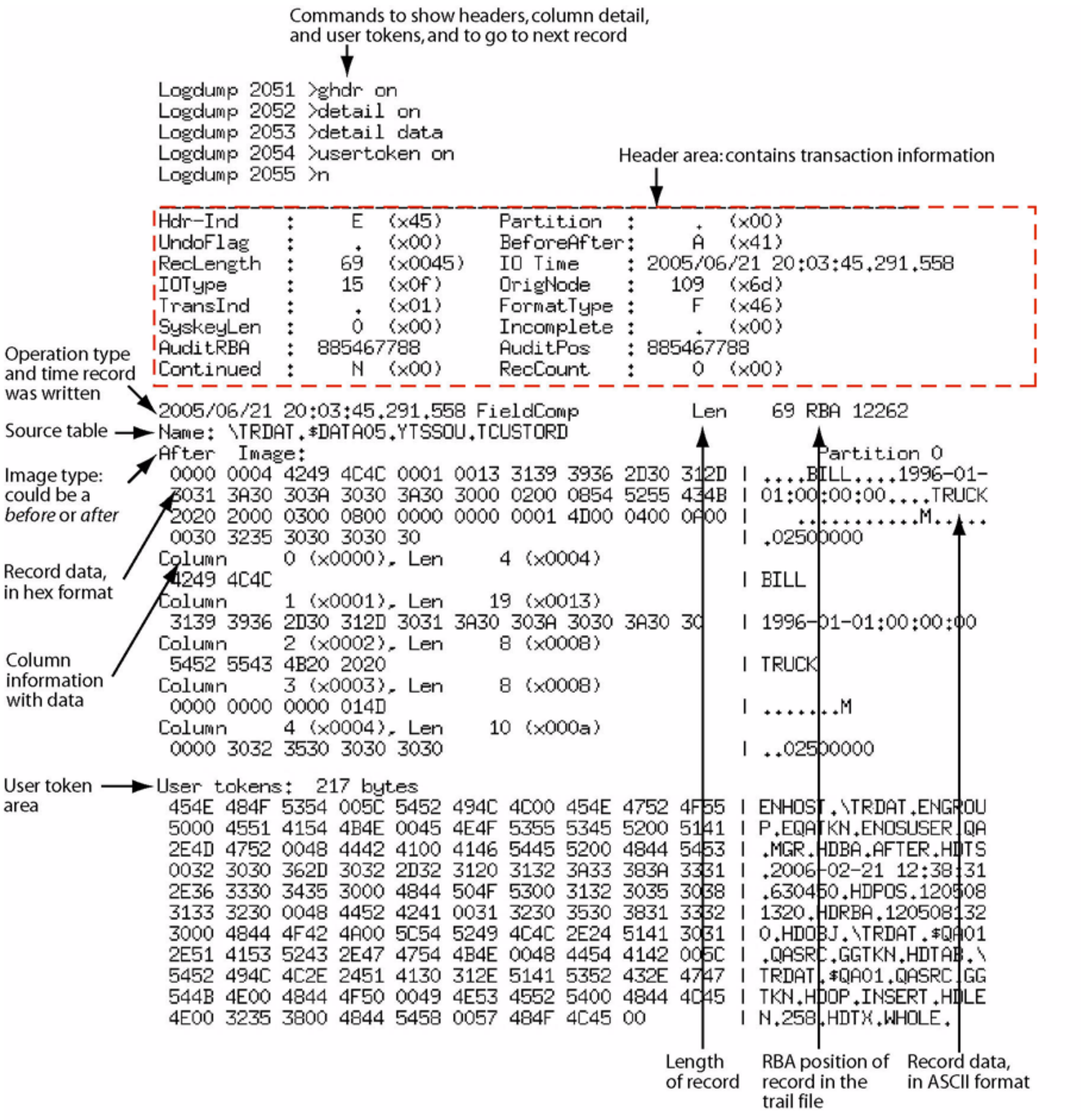
   ```
   open $data01.glogggl.aa000000
   ```

2. To go to the first record and then move through records in sequence:

   ```
   Logdump 7> NEXT
   ```
   (or just type an N)

A GoldenGate trail record looks similar to the one in the following illustration, depending on what views are activated and what type of record it is. In this case, commands were issued to show the header portion of the record, to show column-level detail, and to show user tokens.

Figure 34 Sample trail record as viewed in Logdump on a NonStop system


Commands to show headers, column detail, and user tokens, and to go to next record
Header area: contains transaction information
Operation type and time record was written
Source table
Image type: could be a *before* or *after*
Record data, in hex format
Column information with data
User token area
Length of record
RBA position of record in the trail file
Record data, in ASCII format

```
Logdump 2051 >ghdr on
Logdump 2052 >detail on
Logdump 2053 >detail data
Logdump 2054 >usertoken on
Logdump 2055 >n

Hdr-Ind     :     E  (x45)     Partition  :     .  (x00)
UndoFlag    :     .  (x00)     BeforeAfter:     A  (x41)
RecLength   :    69  (x0045)   IO Time    : 2005/06/21 20:03:45.291.558
IOType      :    15  (x0f)     OrigNode   :   109  (x6d)
TransInd    :     .  (x01)     FormatType :     F  (x46)
SyskeyLen   :     0  (x00)     Incomplete :     .  (x00)
AuditRBA    :  885467788       AuditPos   : 885467788
Continued   :     N  (x00)     RecCount   :     0  (x00)

2005/06/21 20:03:45.291.558 FieldComp           Len    69 RBA 12262
Name: \TRDAT.$DATA05.YTSSOU.TCUSTORD
After  Image:                                              Partition 0
 0000 0004 4249 4C4C 0001 0013 3139 3936 2D30 312D  | ....BILL....1996-01-
 3031 3A30 303A 3030 3A30 3000 0200 0854 5255 434B  | 01:00:00:00....TRUCK
 2020 2000 0300 0800 0000 0000 0001 4D00 0400 0A00  |    ...........M.....
 0030 3235 3030 3030 30                             | .02500000
Column     0 (x0000), Len     4 (x0004)
 4249 4C4C                                          | BILL
Column     1 (x0001), Len    19 (x0013)
 3139 3936 2D30 312D 3031 3A30 303A 3030 3A30 30    | 1996-01-01:00:00:00
Column     2 (x0002), Len     8 (x0008)
 5452 5543 4B20 2020                                | TRUCK
Column     3 (x0003), Len     8 (x0008)
 0000 0000 0000 014D                                | .......M
Column     4 (x0004), Len    10 (x000a)
 0000 3032 3530 3030 3030                           | ..02500000

User tokens:  217 bytes
 454E 484F 5354 005C 5452 494C 4C00 454E 4752 4F55  | ENHOST.\TRDAT.ENGROU
 5000 4551 4154 4B4E 0045 4E4F 5355 5345 5200 5141  | P.EQATKN.ENOSUSER.QA
 2E4D 4752 0048 4442 4100 4146 5445 5200 4844 5453  | .MGR.HDBA.AFTER.HDTS
 0032 3030 362D 3032 2D32 3120 3132 3A33 383A 3331  | .2006-02-21 12:38:31
 2E36 3330 3435 3000 4844 504F 5300 3132 3035 3038  | .630450.HDPOS.120508
 3133 3230 0048 4452 4241 0031 3230 3530 3831 3332  | 1320.HDRBA.120508132
 3000 4844 4F42 4A00 5C54 5249 4C4C 2E24 5141 3031  | 0.HDOBJ.\TRDAT.$QA01
 2E51 4153 5243 2E47 4754 4B4E 0048 4454 4142 005C  | .QASRC.GGTKN.HDTAB.\
 5452 494C 4C2E 2451 4130 312E 5141 5352 432E 4747  | TRDAT.$QA01.QASRC.GG
 544B 4E00 4844 4F50 0049 4E53 4552 5400 4844 4C45  | TKN.HDOP.INSERT.HDLE
 4E00 3235 3800 4844 5458 0057 484F 4C45 00         | N.258.HDTX.WHOLE.
```

Executing basic Logdump tasks

The following are some basic tasks that can be performed with Logdump. For detailed information about the commands shown and other available options, see the alphabetical reference beginning on page 164.

To find the next good record header

```
Logdump 8> SCANFORHEADER
```

(or just type `SFH`)

To find the beginning, middle, and end of a transaction

1. Show headers and detail.

   ```
   Logdump 9> GHDR ON
   Logdump 10> DETAIL ON
   ```

2. Go to the next record.

   ```
   Logdump 11> N
   ```

3. View the TransInd field in the record header. The following tells you where the record is in relation to the transaction.

`TransInd`	`: .`	`(x00)`	First statement in transaction
`TransInd`	`: .`	`(x01)`	Statement in middle of transaction
`TransInd`	`: .`	`(x02)`	Last statement in transaction
`TransInd`	`: .`	`(x03)`	Sole statement in transaction

4. Move through subsequent records by pressing `N`, and refer to the TransInd field to determine where each one is within the transaction. When TransInd is either x02 or x03, the TransInd of the next record should be x00, starting a new transaction.

To scan for the end of a transaction

```
Logdump 20> SCANFORENDTRANSACTION
```

(or just type `SFET`)

The record shown will be the first one in the next transaction. To confirm, the TransInd field should be x00.

To go to a specific RBA in the file

- To go to an RBA anywhere in the file:

  ```
  Logdump 35> POS <rba>
  Logdump 36> N
  ```

  This displays the record located at that RBA.

- To go to the first record in the file:

  ```
  Logdump 37> POS FIRST
  ```

  or...

  ```
  Logdump 37> POS 0
  ```

To filter based on a table or data file name

- To filter out everything except records containing a specific NonStop data file name:

  ```
  Logdump 60> FILTER INCLUDE FILENAME <$volume>.<subvolume>.<file>
  ```

- To filter out everything except records containing a specific table name:

  ```
  Logdump 60> FILTER INCLUDE ANSIINAME <catalog>.<schema>.<table>
  ```

  Now, when you use the N command, you will only see records that satisfy this filter.
- Conversely, to filter out records containing a specific table or file name, but show everything else, use the EXCLUDE option instead of INCLUDE.

To remove the current filter criteria

```
Logdump 62> FILTER CLEAR
```

To filter on multiple conditions

```
Logdump 60> FILTER INCLUDE FILENAME <$volume>.<subvolume>.<file>; FILTER
RECTYPE <record_type>; FILTER MATCH ALL

Or...

Logdump 60> FILTER INCLUDE ANSIINAME <catalog>.<schema>.<table>; FILTER
RECTYPE <record_type>; FILTER MATCH ALL
```

Use MATCH ANY or MATCH ALL depending on whether you want the search to match any or all of the filter conditions, respectively, when multiple conditions are specified. The preceding example filters on a name and record type, typically an operation type such as INSERT.

To count the records in a trail file

```
Logdump 67> COUNT
```

This shows a count summary followed by counts for each table or data file.

To save records to a new trail file

- To save the whole file:

  ```
  Logdump 68> SAVE <file>
  ```

  Where: <file> is the name of the new file.
- To save a subset of records:

  ```
  Logdump 69> SAVE <file> <n> RECORDS
  ```

To close the current file and open the next one in the trail

```
Logdump 70> NEXTTRAIL
```

To keep a log of your session

- To start logging:

  ```
  Logdump 71> LOG TO <filename>.txt
  ```
- To write text to the log:

  ```
  Logdump 72> WRITELOG “<text>”
  ```
- To stop logging:

  ```
  Logdump 73> LOG STOP
  ```

To see the current Logdump environment

```
Logdump 74> ENV
```

This shows which features are enabled, such as filtering and header views, and it shows environment information such as the current trail and position.

To get online command help

```
Logdump 75> HELP
```

To exit Logdump

```
Logdump 100> EXIT
```

Or…

```
Logdump 100> QUIT
```

Evaluating transaction size

Use Logdump's TRANSHIST command in conjunction with other Logdump commands to determine whether or not your applications generate large transactions and to identify their relative size. TRANSHIST causes Logdump to track the size of transactions contained in a trail file or extract file in an internal history table. The transactions are ranked in descending order of size, in bytes. When the history table is full, the smallest transaction is removed to allow a larger transaction to be added to the list.

To use statistics generated by TRANSHIST, issue the following series of commands in Logdump:

1. Use TRANSHIST to set the size of the history table that tracks transaction size. The maximum size is 200 bytes. A value of 0 turns off the tracking.

   ```
   TRANSHIST <n>
   ```

2. Use either the TRANSRECLIMIT or TRANSBYTELIMIT command to set a lower boundary for what is considered a normal sized transaction. These commands prevent normal-sized transactions from being tracked. Eliminating normal-sized transactions reduces the amount of data that must be reviewed.

   ```
   {TRANSBYTELIMIT <n bytes> | TRANSRECLIMIT <n records>}
   ```

3. Use Logdump's COUNT command to display the statistics on transaction size, which appear at the end of the output and look like the following excerpt:

   ```
       Transactions with at least 100 records or 100000 bytes
       2006/02/01 09:31:24.000.000    00:00:00.000, Seq 0, RBA 13101
           Bytes/Trans .....    1168167
           Records/Trans ...       1001
           Files/Trans .....          1
       2006/02/01 09:31:35.000.000    00:00:11.000, Seq 0, RBA 1205292
           Bytes/Trans .....    1168167
           Records/Trans ...       1001
           Files/Trans .....          1
   ```

Logdump scans the file(s) and reports the information.

4. Use Logdump's POSITION <RBA> command to go to each RBA listed in the COUNT output to find out the name of the table that generated the transaction. You can group these tables into their own processing group so that they do not affect processing of other tables that generate normal sized transactions.

Maintaining command history

On NonStop systems, command history is stored in a file named logduhst. The file is created in the home location of the user who first started Logdump.

When Logdump starts up, it looks for the history file in:

- The default $vol.subvol.

If the file exists, Logdump loads the command history into a buffer. The command history buffer holds 400 commands. Upon termination of the Logdump session, the session's history is appended to the file.

Logdump command reference guide

This is the reference guide for Logdump commands.

Logdump command summary

The following are category summaries of the Logdump commands.

Table 8 Working with files

Commands	Description
LOG	Writes a session log.
NEXTTRAIL	Closes the current file and opens the next file in the trail sequence.
OPEN	Opens a trail file or extract file.
POSITION	Sets the read position in the file.
SAVE	Writes record data to another file.
WRITELOG	Writes text to a session log.
VOLUME	Sets the default volume or subvolume.

Table 9 Viewing information

Command	Description
COUNT	Displays record count information.
FILES	Displays file names in the currentsubvolume.
ENV	Displays current Logdump settings.
NOTIFY	Displays the number of records scanned, the trail position, and the record timestamp at specified intervals when using COUNT and records are being suppressed from display through filtering options.
SHOW	Displays internal information such as files that are open, the current Logdump environment, a list of GoldenGate record types, and current filter settings.
TIME	Displays the current time in local and GMT formats.

Table 10 Selecting data and records

Command	Description
DUMP	Displays the specified number of bytes of data from the current position in the file.
FILTER	Filters the display of records.
NEXT	Displays the next record(s) in the file.
SCANFORENDTRANSACTION	Finds a record that is the last record of, or the only record in, a transaction, and then displays the first record of the next transaction.
SCANFORHEADER	Finds the start of the next record header.
SCANFORRBA	Finds a specific relative byte address.
SCANFORTIME	Finds the next record with a specific timestamp.
SCANFORTYPE	Finds the next record of a specific type.
SKIP	Skips a specified number of records.

Table 11 Making conversions

Command	Description
COMPUTETIMESTAMP	Converts a datetime string to a Julian timestamp.
CTIME	Converts a C timestamp to an ASCII timestamp.
DECRYPT	Decrypts data before displaying it in Logdump.
ENCRYPT	Encrypts file data.
ESBLOCK	Displays NonStop entry-sequenced syskeys as a block number and record number.
INTERPRETINTERVAL	Displays a 64-bit Julian interval as days-hh:mm:ss:ms:us.
INTERPRETTIMESTAMP	Displays a 64-bit Julian timestamp in ASCII format.

Table 12 Controlling the Logdump environment

Command	Description
DETAIL	Controls the display of detailed record information.
GHDR	Controls the display of header information.
HEADERTOKEN	Controls the display of header token indicators.
RECLEN	Sets the maximum data output length.
SCANSCROLLING	Controls whether a count notification displays on one line or multiple lines.
TIMEOFFSET	Sets the time offset from GMT.
TMFBEFOREIMAGE	Controls whether or not the before image is displayed for update operations from TMF audit.
TRAILFORMAT	Sets the trail format to the old version (pre-GoldenGate 6.0) or the new version.
TRANSBYTELIMIT	Sets a byte-count threshold for what is defined as a normal-sized transaction.
TRANSHIST	Sets the size of the transaction history table that is used for tracking transaction size.

Table 12 Controlling the Logdump environment (continued)

Command	Description
TRANSRECLIMIT	Sets a record-count threshold for what is defined as a normal-sized transaction.
USERTOKEN	Controls the display of user token data.

Table 13 Miscellaneous commands

Command	Description
DEBUG	Turns on Logdump debugging.
EXIT	Exits Logdump.
FC	Edits a previous command.
HELP	Shows syntax for Logdump commands.
HISTORY	Lists previously issued commands.
OBEY	Executes a series of commands stored in a file.
X	Executes a program from within Logdump.

COMPUTETIMESTAMP

Use COMPUTETIMESTAMP to convert a datetime string to Julian format.

Default None

Syntax
```
COMPUTETIMESTAMP <datetime string>
```

Argument	Description
`<datetime string>`	A datetime string in the format of: [[yy]yy-mm-dd] [hh[:mm][:ss]]

Example
```
COMPUTETIMESTAMP 2005-01-01 12:00:00
```
This returns the following:
```
2005-01-01 12:00:00 is JulianTimestamp 211971340800000000
```

COUNT

Use COUNT to produce a record count summary and other information related to the amount of data in the file. The basic output, without options, shows the following:

- The RBA where the count began
- The number of records in the file
- The total data bytes and average bytes per record
- Information about the operation types
- Information about the transactions

When the DETAIL command is issued prior to issuing COUNT, the information includes a count for each tableor data file. COUNT options allow you to show table detail without using the DETAIL command first, set a start and end time for the count, filter the count for a table, data file, trail file, or extract file, and specify a time interval for counts.

For arguments that take a time string, use the following format:

[[yy]yy-mm-dd] [hh[:mm][:ss]]

Default Produce a count summary of all records.

Syntax

```
COUNT
[, DETAIL]
[, END[TIME] <time_string>]
[, FILE <specification>]
[, INT[ERVAL] <minutes>]
[, LOG] <wildcard>]
[, START[TIME] <time_string>]
```

Argument	Description
`DETAIL`	Adds a count for each table or data file that was processed by Extract to the summary count. The information includes the total and average number of data bytes and information about the operations that were performed. This data can also be obtained by using the DETAIL command before issuing COUNT.
`END[TIME] <time_string>`	Stops the count with the last record written at the specified time.
`FILE <specification>`	Specifies the count to be generated for the specified table or data file or group of names designated with a wildcard (*).
`INT[ERVAL] <minutes>`	Displays statistics for total bytes, average bytes, and number of each type of operation that occurred within a specified interval of time, in minutes. Then it displays the totals for those statistics.
`LOG <wildcard>`	Produces a count for multiple trail or extract files specified with a wildcard.
`START[TIME] <time_string>`	Begins the count with the first record written at the specified time.

Example 1 `COUNT START 2006-01-11 12:00:00 , END 2006-01-12 12:00:00`

Example 2 `COUNT INTERVAL 4`

This displays something similar to the following (individual table or data file count has been truncated due to space constraints):

```
LogTrail \GGQA.$QA01.QADAT.LS000000 has 29656 records
Total Data Bytes           3561022
  Avg Bytes/Record             120
Delete                          50
Insert                       21221
Update                        8379
GSSPurgedata                     6
Before Images                   50
After Images                 29606

Average of 3621 Transactions
    Bytes/Trans .....       1376
    Records/Trans ...          8
    Files/Trans .....          1

\GGQA.$QA01.QAESRC.ACCTS                         Partition 0
Total Data Bytes            286414
  Avg Bytes/Record             142
Delete                          17
Insert                        2000
Before Images                   17
After Images                  2000

\GGQA.$QA01.QAESRC.ACCTN                         Partition 0
Total Data Bytes            281700
  Avg Bytes/Record             100
Delete                          17
Insert                        2000
Update                         800
Before Images                   17
After Images                  2800
```

Example 3 `COUNT LOG ls*`

This produces a count for all files whose names begin with LS. (Individual table or data file count has been truncated due to space constraints.)

```
Current LogTrail is \GGQA.$QA01.QADAT.LS000000
Bad record found at RBA 5287, format 5.50)
 2A56 623F                                       | *Vb?
LogTrail \GGQA.$QA01.QADAT.LS000000 has 33 records
LogTrail \GGQA.$QA01.QADAT.LS000000 closed
Current LogTrail is \GGQA.$QA01.QADAT.LS000001
LogTrail \GGQA.$QA01.QADAT.LS000001 has 99 records
LogTrail \GGQA.$QA01.QADAT.LS000001 closed
Current LogTrail is \GGQA.$QA01.QADAT.LS000002
LogTrail \GGQA.$QA01.QADAT.LS000002 has 0 records
LogTrail \GGQA.$QA01.QADAT.LS000002 closed
Current LogTrail is \GGQA.$QA01.QADAT.LS000003
LogTrail \GGQA.$QA01.QADAT.LS000003 has 0 records
LogTrail \GGQA.$QA01.QADAT.LS000003 closed
LogTrail \GGQA.$QA01.QADAT.LS* has 132 records

Total Data Bytes              9468
  Avg Bytes/Record              71
Insert                         132
After Images                   132

Average of 4 Transactions
    Bytes/Trans .....         3951
    Records/Trans ...           33
    Files/Trans .....            3

QAHRTS.JOBS                                          Partition 4
Total Data Bytes              5220
  Avg Bytes/Record              68
Insert                          76
After Images                    76
```

Example 4

```
COUNT DETAIL
```

This produces something similar to the following:

```
LogTrail \GGQA.$QA01.QADAT.LS000000 has 29656 records
Total Data Bytes          3561022
  Avg Bytes/Record            120
Delete                         50
Insert                      21221
Update                       8379
GSSPurgedata                    6
Before Images                  50
After Images                29606

Average of 3621 Transactions
    Bytes/Trans .....       1376
    Records/Trans ...          8
    Files/Trans .....          1

\GGQA.$QA01.QAESRC.ACCTS                         Partition 0
Total Data Bytes           286414
  Avg Bytes/Record            142
Delete                         17
Insert                       2000
Before Images                  17
After Images                 2000
```

CTIME

Use CTIME to convert a C timestamp to an ASCII timestamp.

Default None

Syntax `CTIME <C timestamp string>`

Example `CTIME 1109823330`

This returns the following:

```
timestamp  = 1109823330 (0x42268f62)
localtime  = Wed Mar  2 20:15:30 2005
gmtime     = Thu Mar  3 04:15:30 2005
```

DEBUG

Use DEBUG to run debugging for Logdump. Use this command with the guidance of a GoldenGate support analyst.

Default Disabled

Syntax `DEBUG`

DECRYPT

Use DECRYPT to decrypt data that was encrypted with GoldenGate trail encryption, so that it can be viewed with Logdump.

Default `OFF`

Syntax `DECRYPT {ON | OFF}`

DETAIL

Use DETAIL to include additional information in the Logdump output. By default, Logdump only shows the hex and ASCII representation of the record.

Without options, DETAIL displays the status of record detail (ON or OFF). Options do the following:

- DETAIL ON displays a list of columns that includes the column ID, length, and value in hex and ASCII.
- DATA adds hex and ASCII data values to the column list.
- DETAIL OFF turns off detailed display.

For an illustration of how DETAIL output looks, see Figure 34 on page 160.

DETAIL can be shortened to DET.

Default Display a column list

Syntax `DETAIL {ON | OFF | DATA}`

Argument	Description
`ON`	Shows detailed column information.
`OFF`	Suppresses detailed column information.
`DATA`	Adds the hex and ASCII data values to the column information.

DUMP

Use DUMP to display a HEX/ASCII or HEX/EBCDIC dump of the specified number of bytes from the open trail or extract file, starting at the current RBA.

DUMP does not work when reading TMF audit trails, because I/O to the TMF trails is done by TMFARLIB.

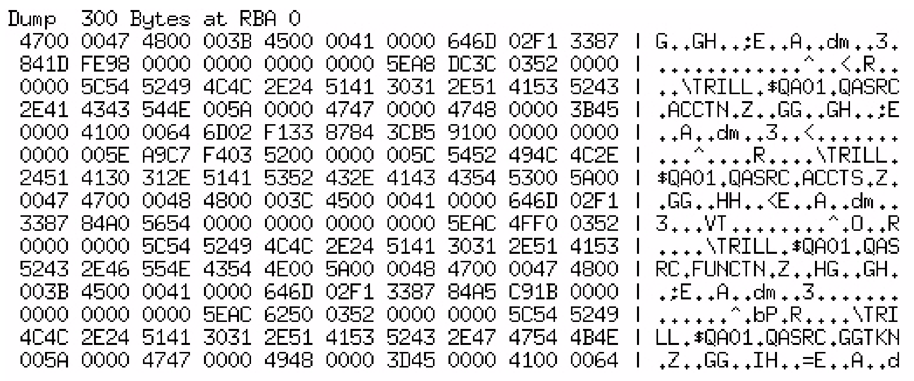
Default 256

Syntax `DUMP <bytes>`

Argument	Description
`<bytes>`	The number of bytes forward to display. Valid values are from 1 through 28672.

Example `DUMP 300`

This produces something similar to the following:



```
Dump  300 Bytes at RBA 0
 4700 0047 4800 003B 4500 0041 0000 646D 02F1 3387 | G..GH..;E..A..dm..3.
 841D FE98 0000 0000 0000 0000 5EA8 DC3C 0352 0000 | ............^..<.R..
 0000 5C54 5249 4C4C 2E24 5141 3031 2E51 4153 5243 | ..\TRILL.$QA01.QASRC
 2E41 4343 544E 005A 0000 4747 0000 4748 0000 3B45 | .ACCTN.Z..GG..GH..;E
 0000 4100 0064 6D02 F133 8784 3CB5 9100 0000 0000 | ..A..dm..3..<.......
 0000 005E A9C7 F403 5200 0000 005C 5452 494C 4C2E | ...^....R....\TRILL.
 2451 4130 312E 5141 5352 432E 4143 4354 5300 5A00 | $QA01.QASRC.ACCTS.Z.
 0047 4700 0048 4800 003C 4500 0041 0000 646D 02F1 | .GG..HH..<E..A..dm..
 3387 84A0 5654 0000 0000 0000 0000 5EAC 4FF0 0352 | 3...VT........^.O..R
 0000 0000 5C54 5249 4C4C 2E24 5141 3031 2E51 4153 | ....\TRILL.$QA01.QAS
 5243 2E46 554E 4354 4E00 5A00 0048 4700 0047 4800 | RC.FUNCTN.Z..HG..GH.
 003B 4500 0041 0000 646D 02F1 3387 84A5 C91B 0000 | .;E..A..dm..3.......
 0000 0000 0000 5EAC 6250 0352 0000 0000 5C54 5249 | ......^.bP.R....\TRI
 4C4C 2E24 5141 3031 2E51 4153 5243 2E47 4754 4B4E | LL.$QA01.QASRC.GGTKN
 005A 0000 4747 0000 4948 0000 3D45 0000 4100 0064 | .Z..GG..IH..=E..A..d
```

ENCRYPT

Use ENCRYPT to encrypt text supplied as an argument. The encryption method is 256-key byte substitution. The results are printed to screen.

Default None

Syntax `ENCRYPT <text>`

Example `ENCRYPT 123456789`

This produces the following:

```
Before
 3132 3334 3536 3738 39 | 123456789
After
 EF2E C1DC E4A7 68B4 14 | ......h..
```

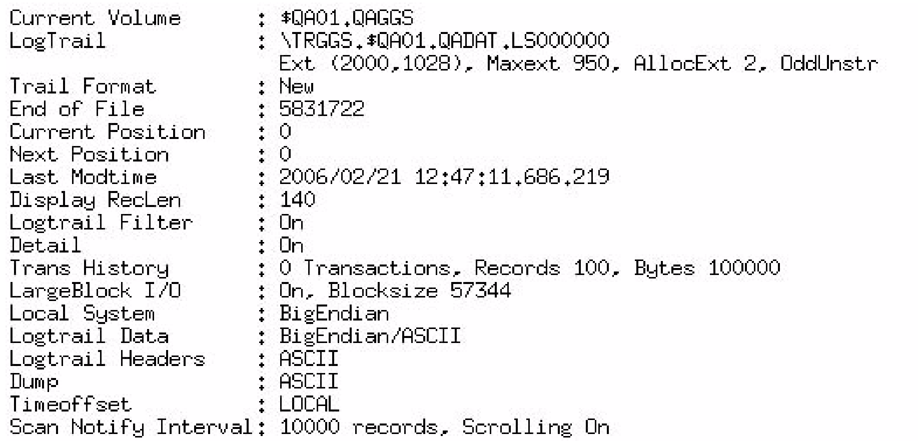
ENV

Use ENV to show current Logdump settings.

Default None

Syntax `ENV`

Example The following shows typical ENV settings.



```
Current Volume      : $QA01.QAGGS
LogTrail            : \TRGGS.$QA01.QADAT.LS000000
                      Ext (2000,1028), Maxext 950, AllocExt 2, OddUnstr
Trail Format        : New
End of File         : 5831722
Current Position    : 0
Next Position       : 0
Last Modtime        : 2006/02/21 12:47:11.686.219
Display RecLen      : 140
Logtrail Filter     : On
Detail              : On
Trans History       : 0 Transactions, Records 100, Bytes 100000
LargeBlock I/O      : On, Blocksize 57344
Local System        : BigEndian
Logtrail Data       : BigEndian/ASCII
Logtrail Headers    : ASCII
Dump                : ASCII
Timeoffset          : LOCAL
Scan Notify Interval: 10000 records, Scrolling On
```

ESBLOCK

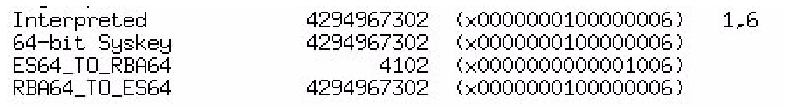
Use ESBLOCK for debugging on a NonStop system. It displays an entry-sequenced syskey as a block number and record number.

Default None

Syntax `ESBLOCK <entry-sequenced RBA>`

Example `ESBLOCK 4294967302`

This produces the following. It shows that the syskey value 4294967302 evaluates to block 1, record 6.



```
Interpreted            4294967302  (x0000000100000006)   1,6
64-bit Syskey          4294967302  (x0000000100000006)
ES64_TO_RBA64                4102  (x0000000000001006)
RBA64_TO_ES64          4294967302  (x0000000100000006)
```

EXIT

Use EXIT to exit Logdump and terminate the process. An alias for EXIT is QUIT.

Default None

Syntax `EXIT`

FC

Use FC to edit a previously issued Logdump command and then execute it again. Previous commands are stored in the memory buffer and can be displayed by issuing the HISTORY command (see page 194). Issuing FC without arguments executes the most recently used command. By using options, you can retrieve a specific command by specifying its line number or a text substring.

Using the editor

The FC command displays the command and then opens an editor with a prompt containing a blank line starting with two dots. To edit a command, use the space bar to position the cursor beneath the character where you want the change to begin, and then enter one of the following arguments. Arguments are not case-sensitive and can be combined.

Argument	Description

`i <text>`	Inserts text. For example:
	`Logdump 24> fc 9`
	`> count`
	`..     i detail`
	`count detail`

`r <text>`	Replaces text. For example:
	`Logdump 25> fc 10`
	`> timeoffset local`
	`..           rgmt`
	`timeoffset gmt`

`d`	Deletes a character. To delete multiple characters, enter a `d` for each one. For example:
	`Logdump 26> fc 11`
	`> scanforrrbba`
	`..        dd`
	`scanforrba`

`<replacement text>`	Replaces the displayed command with the text that you enter on a one-for-one basis. For example:
	`Logdump 26> fc 10`
	`> scanforrba 107`
	`..           127`
	`scanforrba 127`

To execute the command, press Enter twice, once to exit the editor and once to issue the command. To cancel an edit, type a forward slash (/) twice.

Default Execute the most recent command again

Syntax `FC [<n> | -<n> | <string>]`

Argument	Description
`<n>`	Returns the specified command line. Each Logdump command line is sequenced, beginning with 1 at the start of the session.
`-<n>`	Returns the command that was issued <n> lines before the current line.
`<string>`	Returns the last command that starts with the specified text string.

Example 1 `FC 9`

Example 2 `FC -3`

Example 3 `FC sca`

FILEHEADER

Use FILEHEADER to display the contents of the header of the currently open trail file.

The file header is stored as a record at the beginning of a trail file preceding the data records. The information that is stored in the trail header provides enough information about the records to enable a GoldenGate process to determine whether the records are in a format that the current version of GoldenGate supports.

The trail header fields are stored as tokens, where the token format remains the same across all versions of GoldenGate. If a version of GoldenGate does not support any given token, that token is ignored. Depracated tokens are assigned a default value to preserve compatibility with previous versions of GoldenGate.

The current FILEHEADER command applies globally to the Logdump session, until a different FILEHEADER command is issued.

To view the file header

1. Position to the beginning of the trail file with the following Logdump command.

   ```
   pos 0
   ```

2. Issue the following Logdump command to see the first record of the file, the one that contains the file header.

   ```
   next
   ```

To retrieve the file header tokens

To retrieve file header values as input parameters, use the @GETENV function with the GGFILEHEADER option. See the *Reference Guide*.

NOTE The Logdump command HEADERTOKEN also shows trail tokens, but it shows a brief summary of each one. FILEHEADER shows actual token values.

Table 14 GoldenGate file header tokens

Token/subtoken	Data Type	Description
TrailInfo		**Information about the trail file.**
Signature	UINT32	Internal use.
Compatibility	UINT16	The version of the trail. The compatibility level of the GoldenGate software must be greater than, or equal to, that of the trail file for a process to be able to read the trail file. Current valid values are 0 or 1.

Table 14 GoldenGate file header tokens

Token/subtoken	Data Type	Description
CharSet	INT32	The global character set of the trail file, as defined in the parameter file or the default value. For example: WCP1252-1 -3 indicates the system default.
CreationTime	Timestamp	The time that the trail file was created, in local GMT Julian time, INT 64.
URI	UString	The universal resource identifier of the process that created the trail file, in the format of: <host_name>:<dir>:[:<dir>][:<dir_n>]<group_name> **Where:** ◆ host_name is the name of the server that hosts the process ◆ dir is a subdirectory of the GoldenGate installation path. ◆ group_name is the name of the process group that is linked with the process. **Example**: sys1:home:oracle:v9.5:extora Shows where the trail was processed and by which process. This includes a history of previous runs.
URIHistory	UString Vector	List of the URIs of processes that wrote to the trail file before the current process. ◆ For a primary Extract, this field is empty. ◆ For a data pump, this field is URIHistory + URI of the input trail file.
FileName	UString	Name of the trail file. Can be absolute or relative path, with forward or backward slash depending on the filesystem.
MultiPart	Boolean	True/false flag indicating whether the trail file is a single file (such as one created for a batch run) or a sequentially numbered file that is part of a trail for online, continuous processing. If false, the SeqNum subtoken is not valid.
SeqNum	UINT32	The sequence number of the file in the trail, if MultiPart is true. Invalid if multipart is false. The value is the numerical sequence number, without any zero padding.
FileSize	UINT642	Size of the trail file. Value is NULL until the trail file is completed. Non-NULL values are in bytes.
FirstRecordCSN	CSN	The commit sequence number (CSN) of the first record in the trail file.Value is NULL until the trail file is completed.

Table 14 GoldenGate file header tokens

Token/subtoken	Data Type	Description
LastRecordCSN	CSN	The commit sequence number (CSN) of the last record in the trail file.Value is NULL until the trail file is completed.
FirstRecordIOTime	Timestamp	The time that the first record in the trail file was written.Value is NULL until the trail file is completed.
LastRecordIOTime	Timestamp	The time that the last record in the trail file was written.Value is NULL until the trail file is completed.
MachineInfo		**Information about the local host of the trail file.**
SysName	UString	The name of the operating system, for example: SunOX Linux Microsoft Windows
NodeName	UString	The name of the machine, for example sys1.
Release	UString	The release level of the operating system, for example: 5.10 2.6.9-11.ELsmp 2000 Advanced Server
Version	UString	The version of the operating system, for example: s10_69 #1 SMP Fri Feb 24 16:56:28 EST 2006 5.00.2195 Service Pack 4
Hardware	UString	The hardware type of the processor, for example: sun4u x86_64 x86
DatabaseInfo		**Information about the database that produced the data in the trail file.**

Table 14 GoldenGate file header tokens

Token/subtoken	Data Type	Description

Vendor	UINT16	The name of the database vendor. Can be one of:
		`DB2 UDB`
		`DB2 ZOS`
		`CTREE`
		`INGRES`
		`MSSQL`
		`MYSQL`
		`ORACLE`
		`SQLMX`
		`SYBASE`
		`TERADATA`
		`TIMESTEN`
		`NONSTOP`
		`ENSCRIBE`
		`MSACCESS`
		`ODBC`

Name	UString	The name of the database, for example findb.
Instance	UString	The name of the database instance, if applicable to the database type, for example ORA1022A.
Charset	INT32	The character set of the database. Currently, the valid value is -1 (unknown). (For some databases, this will be empty.)
MajorVersion	UINT16	The major version of the database.
MinorVersion	UINT16	The minor version of the database.
VerString	UString	The maintenance (patch) level of the database.
ClientCharset	INT32	The character set of the database client. Currently, the valid value is -1 (unknown). (For some databases, this will be empty.)
ClientVerString	UString	The maintenance (patch) level of the database client. (For some databases, this will be empty.)
ProducerInfo		**Information about the GoldenGate process that created the trail file.**
Name	UString	The group name that is associated with the process.

Table 14 GoldenGate file header tokens

Token/subtoken	Data Type	Description
DataSource	UINT16	The data source that was read by the process. Can be one of: ◆ DS_EXTRACT_TRAILS (source was a GoldenGate extract file, populated with change data) ◆ DS_LOG_TABLE (source was a GoldenGate log table, used for trigger-based extraction) ◆ DS_DATABASE (source was a direct select from database table written to a trail, used for SOURCEISTABLE-driven initial load) ◆ DS_TRAN_LOGS (source was the database transaction log) ◆ DS_INITIAL_DATA_LOAD (source was Extract; data taken directly from source tables) ◆ DS_VAM_EXTRACT (source was a vendor access module) ◆ DS_VAM_TWO_PHASE_COMMIT (source was a VAM trail)
MajorVersion	UINT16	The major version of the process (xx).
MinorVersion	UINT16	The minor version of the process (xx.xx).
MaintenanceLevel	UINT16	The maintenance version of the process (xx.xx.xx).
PatchLevel	UINT16	The patch version of the process (xx.xx.xx.xx).
BuildNumber	UINT16	The build number of the process.
VerString	UString	The version string of the process. For example: 9.5.1.17A not for production
ContinuityInfo		**Contains recovery information that is carried over from the previous trail file in the sequence.**
RecoveryMode	UINT16	Internal use
LastCompletedCSN	CSN	Internal use
LastCompletedXids	Xid	Internal use
LastSCN	CSN	Internal use
LastXid	Xid	Internal use

Default OFF

Syntax `FILEHEADER {ON | OFF | DETAIL}`

Argument	**Description**
`ON`	Enables the display of the file header, showing the main header tokens.
`OFF`	Disables the display of the file header.
`DETAIL`	Provides detailed information that includes the sub-tokens.

Example 1

```
FILEHEADER ON
TokenID x46 'F' Record Header    Info x00  Length  587
TokenID x30 '0' TrailInfo        Info x00  Length  303
TokenID x31 '1' MachineInfo      Info x00  Length  103
TokenID x32 '2' DatabaseInfo     Info x00  Length   88
TokenID x33 '3' ProducerInfo     Info x00  Length   85
TokenID x34 '4' ContinunityInfo  Info x00  Length    4
TokenID x5a 'Z' Record Trailer   Info x00  Length  587

2008/07/18 13:39:18.951.346 FileHeader           Len   587 RBA 0
Name: *FileHeader*
 3000 012f 3000 0008 660d 0a71 3100 0006 0001 3200 | 0../0...f..q1.....2.
 0008 0000 0016 3300 000c 02f1 7834 eac7 7f3f 3400 | ......3.....x4...?4.
 0037 0031 7572 693a 7465 6c6c 7572 6961 6e3a 3a68 | .7.1uri:tellurian::h
 6f6d 653a 6d63 6361 7267 6172 3a67 6773 3a67 6773 | ome:mccargar:ggs:ggs
 4f72 6163 6c65 3a73 6f75 7263 6536 0000 1700 112e | Oracle:source6......
 2f64 6972 6461 742f 6572 3030 3030 3030 3700 0005 | /dirdat/er0000007...
 0138 0000 0800 01e2 4039 0000 0c00 0000 0000 001d | .8......@9..........

GroupID x30 '0' TrailInfo        Info x00  Length  303
 3000 012f 3000 0008 660d 0a71 3100 0006 0001 3200 | 0../0...f..q1.....2.
 0008 0000 0016 3300 000c 02f1 7834 eac7 7f3f 3400 | ......3.....x4...?4.
 0037 0031 7572 693a 7465 6c6c 7572 6961 6e3a 3a68 | .7.1uri:tellurian::h
 6f6d 653a 6d63 6361 7267 6172 3a67 6773 3a67 6773 | ome:mccargar:ggs:ggs
 4f72 6163 6c65 3a73 6f75 7263 6536 0000 1700 112e | Oracle:source6......
 2f64 6972 6461 742f 6572 3030 3030 3030 3700 0005 | /dirdat/er0000007...
 0138 0000 0800 01e2 4039 0000 0c00 0000 0000 001d | .8......@9..........
 a33b 0000 450a 3634 3136 3138 3936 3932 0000 0000 | .;..E.6416189692....
 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 | ....................
 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 | ....................
 0000 0000 0000 0000 0000 3aff 0045 0000 0000 0000 | ..........:..E......
 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 | ....................
 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 | ....................
 0000 0000 0000 0000 0000 0000 0000 0000 0000 003d | ...................=
 0000 0c02 f178 34eb 556a 403c ff00 0c00 0000 0000 | .....x4.Uj@<........
 0000 00                                           | ...

GroupID x31 '1' MachineInfo      Info x00  Length  103
 3100 0067 3000 000b 0005 4c69 6e75 7831 0000 0f00 | 1..g0.....Linux1....
 0974 656c 6c75 7269 616e 3200 0014 000e 322e 362e | .tellurian2.....2.6.
 392d 3131 2e45 4c73 6d70 3300 0029 0023 2331 2053 | 9-11.ELsmp3..).##1 S
 4d50 2046 7269 204d 6179 2032 3020 3138 3a32 353a | MP Fri May 2018:25:
 3330 2045 4454 2032 3030 3534 0000 0c00 0678 3836 | 30 EDT 20054.....x86
 5f36 34                                           | _64

GroupID x32 '2' DatabaseInfo     Info x00  Length   88
 3200 0058 3000 0006 0007 3100 000e 0008 4f52 4131 | 2..X0.....1.....ORA1
 3032 3241 3200 000e 0008 6f72 6131 3032 3261 3300 | 022A2.....ora1022a3.
 0008 ffff ffff 3400 0006 0000 3500 0006 0000 3600 | ......4.....5.....6.
 0006 0000 3700 0008 ffff ffff 3800 0010 000a 3130 | ....7.......8.....10
 2e32 2e30 2e32 2e30                               | .2.0.2.0
```

```
GroupID x33 '3' ProducerInfo     Info x00  Length   85
 3300 0055 3000 000a 0004 4546 4152 3100 0006 0003 | 3..U0.....EFAR1.....
 3200 0006 0000 3300 0006 0000 3400 0006 0000 3500 | 2.....3.....4.....5.
 0006 0000 3600 0006 0017 3700 0023 001d 5665 7273 | ....6.....7..#..Vers
 696f 6e20 5374 2e20 416e 6472 6577 7320 4275 696c | ion St. Andrews Buil
 6420 3032 33                                      | d 023

GroupID x34 '4' ContinunityInfo  Info x00  Length    4
 3400 0004                                         | 4...
```

Example 2 FILEHEADER DETAIL

```
TokenID x46 'F' Record Header    Info x00  Length  587
TokenID x30 '0' TrailInfo        Info x00  Length  303
TokenID x31 '1' MachineInfo      Info x00  Length  103
TokenID x32 '2' DatabaseInfo     Info x00  Length   88
TokenID x33 '3' ProducerInfo     Info x00  Length   85
TokenID x34 '4' ContinunityInfo  Info x00  Length    4
TokenID x5a 'Z' Record Trailer   Info x00  Length  587

2008/07/18 13:40:26.034.631 FileHeader           Len   587 RBA 0
Name: *FileHeader*
 3000 012f 3000 0008 660d 0a71 3100 0006 0001 3200 | 0../0...f..q1.....2.
 0008 0000 0016 3300 000c 02f1 7834 eac7 7f3f 3400 | ......3.....x4...?4.
 0037 0031 7572 693a 7465 6c6c 7572 6961 6e3a 3a68 | .7.1uri:tellurian::h
 6f6d 653a 6d63 6361 7267 6172 3a67 6773 3a67 6773 | ome:mccargar:ggs:ggs
 4f72 6163 6c65 3a73 6f75 7263 6536 0000 1700 112e | Oracle:source6......
 2f64 6972 6461 742f 6572 3030 3030 3030 3700 0005 | /dirdat/er0000007...
 0138 0000 0800 01e2 4039 0000 0c00 0000 0000 001d | .8......@9..........

GroupID x30 '0' TrailInfo        Info x00  Length  303
 3000 012f 3000 0008 660d 0a71 3100 0006 0001 3200 | 0../0...f..q1.....2.
 0008 0000 0016 3300 000c 02f1 7834 eac7 7f3f 3400 | ......3.....x4...?4.
 0037 0031 7572 693a 7465 6c6c 7572 6961 6e3a 3a68 | .7.1uri:tellurian::h
 6f6d 653a 6d63 6361 7267 6172 3a67 6773 3a67 6773 | ome:mccargar:ggs:ggs
 4f72 6163 6c65 3a73 6f75 7263 6536 0000 1700 112e | Oracle:source6......
 2f64 6972 6461 742f 6572 3030 3030 3030 3700 0005 | /dirdat/er0000007...
 0138 0000 0800 01e2 4039 0000 0c00 0000 0000 001d | .8......@9..........
 a33b 0000 450a 3634 3136 3138 3936 3932 0000 0000 | .;..E.6416189692....
 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 | ....................
 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 | ....................
 0000 0000 0000 0000 0000 3aff 0045 0000 0000 0000 | ..........:..E......
 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 | ....................
 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 | ....................
 0000 0000 0000 0000 0000 0000 0000 0000 0000 003d | ...................=
 0000 0c02 f178 34eb 556a 403c ff00 0c00 0000 0000 | .....x4.Uj@<........
 0000 00                                           | ...
TokenID x30 '0' Signature        Info x00  Length    8
 660d 0a71                                         | f..q
TokenID x31 '1' Compatibility    Info x00  Length    6
 0001                                              | ..
TokenID x32 '2' Charset          Info x00  Length    8
 0000 0016                                         | ....
TokenID x33 '3' CreationTime     Info x00  Length   12
 02f1 7834 eac7 7f3f                               | ..x4...?
TokenID x34 '4' URI              Info x00  Length   55
 0031 7572 693a 7465 6c6c 7572 6961 6e3a 3a68 6f6d | .1uri:tellurian::hom
 653a 6d63 6361 7267 6172 3a67 6773 3a67 6773 4f72 | e:mccargar:ggs:ggsOr
 6163 6c65 3a73 6f75 7263 65                       | acle:source
TokenID x36 '6' Filename         Info x00  Length   23
 0011 2e2f 6469 7264 6174 2f65 7230 3030 3030 30   | .../dirdat/er000000
TokenID x37 '7' MultiPart        Info x00  Length    5
```

```
 01                                               | .
TokenID x38 '8' Seqno            Info x00  Length    8
 0001 e240                                        | ...@
TokenID x39 '9' FileSize         Info x00  Length   12
 0000 0000 0000 1da3                              | ........
TokenID x3b ';' LastCSN          Info x00  Length   69
 0a36 3431 3631 3839 3639 3200 0000 0000 0000 0000 | .6416189692.........
 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 | ....................
 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 | ....................
 0000 0000 00                                     | .....
TokenID x3a ':' FirstCSN         Info xff  Length   69
 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 | ....................
 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 | ....................
 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 | ....................
 0000 0000 00                                     | .....
TokenID x3d '=' LastIOTime       Info x00  Length   12
 02f1 7834 eb55 6a40                              | ..x4.Uj@
TokenID x3c '<' FirstIOTime      Info xff  Length   12
 0000 0000 0000 0000                              | ........

GroupID x31 '1' MachineInfo      Info x00  Length  103
 3100 0067 3000 000b 0005 4c69 6e75 7831 0000 0f00 | 1..g0.....Linux1....
 0974 656c 6c75 7269 616e 3200 0014 000e 322e 362e | .tellurian2.....2.6.
 392d 3131 2e45 4c73 6d70 3300 0029 0023 2331 2053 | 9-11.ELsmp3..).##1 S
 4d50 2046 7269 204d 6179 2032 3020 3138 3a32 353a | MP Fri May 20 18:25:
 3330 2045 4454 2032 3030 3534 0000 0c00 0678 3836 | 30 EDT 20054.....x86
 5f36 34                                          | _64
TokenID x30 '0' Sysname          Info x00  Length   11
 0005 4c69 6e75 78                                | ..Linux
TokenID x31 '1' Nodename         Info x00  Length   15
 0009 7465 6c6c 7572 6961 6e                      | ..tellurian
TokenID x32 '2' Release          Info x00  Length   20
 000e 322e 362e 392d 3131 2e45 4c73 6d70          | ..2.6.9-11.ELsmp
TokenID x33 '3' Version          Info x00  Length   41
 0023 2331 2053 4d50 2046 7269 204d 6179 2032 3020 | .##1 SMP Fri May 20
 3138 3a32 353a 3330 2045 4454 2032 3030 35       | 18:25:30 EDT 2005
TokenID x34 '4' Hardware         Info x00  Length   12
 0006 7838 365f 3634                              | ..x86_64

GroupID x32 '2' DatabaseInfo     Info x00  Length   88
 3200 0058 3000 0006 0007 3100 000e 0008 4f52 4131 |2..X0.....1.....ORA1
 3032 3241 3200 000e 0008 6f72 6131 3032 3261 3300 |022A2.....ora1022a3.
 0008 ffff ffff 3400 0006 0000 3500 0006 0000 3600 |......4.....5.....6.
 0006 0000 3700 0008 ffff ffff 3800 0010 000a 3130 |....7.......8.....10
 2e32 2e30 2e32 2e30                              | .2.0.2.0
TokenID x30 '0' Vendor           Info x00  Length    6
 0007                                             | ..
TokenID x31 '1' Name             Info x00  Length   14
 0008 4f52 4131 3032 3241                         | ..ORA1022A
TokenID x32 '2' Instance         Info x00  Length   14
 0008 6f72 6131 3032 3261                         | ..ora1022a
TokenID x33 '3' Charset          Info x00  Length    8
```

```
 ffff ffff                                      | ....
TokenID x34 '4' MajorVersion     Info x00  Length    6
 0000                                           | ..
TokenID x35 '5' MinorVersion     Info x00  Length    6
 0000                                           | ..
TokenID x36 '6' VerString        Info x00  Length    6
 0000                                           | ..
TokenID x37 '7' ClientCharset    Info x00  Length    8
 ffff ffff                                      | ....
TokenID x38 '8' ClientVerString  Info x00  Length   16
 000a 3130 2e32 2e30 2e32 2e30                  | ..10.2.0.2.0

GroupID x33 '3' ProducerInfo     Info x00  Length   85
 3300 0055 3000 000a 0004 4546 4152 3100 0006 0003 | 3..U0.....EFAR1.....
 3200 0006 0000 3300 0006 0000 3400 0006 0000 3500 | 2.....3.....4.....5.
 0006 0000 3600 0006 0017 3700 0023 001d 5665 7273 | ....6.....7..#..Vers
 696f 6e20 5374 2e20 416e 6472 6577 7320 4275 696c | ion St. Andrews Buil
 6420 3032 33                                   | d 023
TokenID x30 '0' Name             Info x00  Length   10
 0004 4546 4152                                 | ..EFAR
TokenID x31 '1' DataSource       Info x00  Length    6
 0003                                           | ..
TokenID x32 '2' MajorVersion     Info x00  Length    6
 0000                                           | ..
TokenID x33 '3' MinorVersion     Info x00  Length    6
 0000                                           | ..
TokenID x34 '4' MaintLevel       Info x00  Length    6
 0000                                           | ..
TokenID x35 '5' BugFixLevel      Info x00  Length    6
 0000                                           | ..
TokenID x36 '6' BuildNumber      Info x00  Length    6
 0017                                           | ..
TokenID x37 '7' VerString        Info x00  Length   35
 001d 5665 7273 696f 6e20 5374 2e20 416e 6472 6577 | ..Version St.Andrew
 7320 4275 696c 6420 3032 33                    | s Build 023

GroupID x34 '4' ContinunityInfo  Info x00  Length    4
 3400 0004                                      | 4...
```

FILES

Use FILES to display summary file information for files on the local system. The default command displays all files in the curent subvolume. To constrain the display to specific files, you can supply a wildcarded name.

This command can be shortened to FI. An alias for this command is DIR or FILEINFO.

Default Show all files in current subvolume

Syntax `FILES [<subvolume> | <volume.subvolume>]`

Argument	Description
`<subvolume> | <volume.subvolume>`	The name of a subvolume or a wildcard for specific files. Note: On a Windows system, if any file or directory in the specified path contains spaces, the entire path must be enclosed within double quotation marks.

Example `FILES $QAGG.QA01.*`

FILTER

Use FILTER to filter the display based on one or more criteria.

- You can string multiple FILTER commands together, separating each one with a semi-colon, as in:

  ```
  FILTER INCLUDE FILENAME $QA01.QAESRC.ACCTN; FILTER SYSKEY 4294967302;
  FILTER MATCH ALL
  ```

- To avoid unexpected results, avoid stringing filter options together with one FILTER command. For example, the following would be *incorrect*:

  ```
  FILTER INCLUDE FILENAME $QA01.QAESRC.ACCTN; SYSKEY 4294967302
  ```

Without arguments, FILTER displays the current filter status (ON or OFF) and any filter criteria that are in effect.

Comparison operators

For options that take comparison operators, standard operators may be used. These are:

Table 15 Filter option comparison operators

Operator	Example
Equal	= EQ ==
Less than	< LT
Less than or equal	<= LE
Greater than	> GT
Greater than or equal	>= GE

Table 15 Filter option comparison operators

Operator	Example
Not equal	<> NE !=

Note: The absence of an operator implies Equal.

Default Shows current filter settings

Syntax `FILTER [INCLUDE] [EXCLUDE] <filter option>`

Where:

<filter option> can be one of:

```
{
ANSINAME <name> [, <name>] |
AUDITRBA <rba> [<comparison operator>] |
CLEAR {<filter_spec> | ALL} |
CSN | LogCSN [<comparison operator>] [<value>]
ENDTIME <time_string> |
FILENAME <name> [, <name>] |
GGSTOKEN <token name> [<comparison operator>] [<token value>] |
HEX "<hex_string>" [<byte_range>][, "<hex_string>" [<byte_range>]] [...]
|
INT16 <16-bit_integer> |
INT32 <32-bit_integer> |
IOTYPE <operation type> [, <operation type>] |
MATCH {ANY | ALL} |
OFF |
ON |
PROCESS <process_name> |
RBA <byte address> [<comparison operator>] [...] |
RECLEN <length> [<comparison operator>] |
RECTYPE {<type_number> | <type_name>} |
SHOW |
STARTTIME <time_string> |
STRING [BOTH] [B],<text> [<column_range>]
   [[B],<text> [<column_range>]] [...] |
SYSKEY <system key> [<comparison operator>] [...] |
TRANSID <transaction_identifier> |
TRANSIND <indicator> [<comparison operator>] |
TYPE <type> |
UNDOFLAG <type> [<comparison operator>] |
USERTOKEN <token name> [<comparison operator>] [<token value>]
}
```

Argument	Description
`ANSINAME <name> [, <name>]`	Filters based on the ANSI name of a SQL/MX table or a table from a Windows or UNIX source system. For use on NonStop systems. Format for <name> is: Catalog.Schema.Table Up to eight name specifications may be supplied. ANSINAME is case-sensitive. To filter based on the name of a data file, use the FILENAME option.
`AUDITRBA <rba> [<comparison operator>]`	Filters based on the relative byte address of a commit record. For <comparison operator>, see "Comparison operators" on page 187.

`CLEAR {<filter_spec> | ALL}`	Removes filtering criteria.
	◆ ALL removes all filter criteria.
	◆ <filter_spec> removes only the specified criterion. Specify any FILTER option, but not the value. The following example is valid:
	`FILTER CLEAR STRING`
	The following example is not valid:
	`FILTER CLEAR STRING "Denver"`
	An alias for CLEAR is RESET.

`CSN | LogCSN [<comparison operator>] [<value>]`	Filters based on a commit sequence value. For <comparison operator>, see "Comparison operators" on page 187.
`ENDTIME <time_string>`	Ends the filter at the last record written at the specified time. For the time string, use the format of: [[yy]yy-mm-dd] [hh[:mm][:ss]] Example: `ENDTIME 2005-01-12 00:00:00` Can be shortened to ENDTS or END.
`FILENAME <name> [, <name>]`	Filters based on the name of a NonStop data file, or a group of names, with the name format being: <volume>.<subvolume>.<file> <volume>.<subvolume>.<string>* FILENAME is case-sensitive. FILENAME can be shortened to FILE or FI. Up to eight name specifications may be supplied.

Argument	Description
`HEX "<hex_string>" [<byte_range>] [, "<hex_string>" [<byte_range>]] [...]`	Filters based on a hex string and, optionally, a range of columns. To specify a range of columns, use the format of: <start_column>:<end_column> Example: `10:35` This option allows up to eight hex string and column arguments. Hex strings must be enclosed within quotes.
`INCLUDE`	Specifies that the filter will include the information specified with other options in the current FILTER statement. Can be shortened to INC.
`EXCLUDE`	Specifies that the filter will exclude the information specified with other options in the current FILTER statement. Can be shortened to EXC.
`INT16 <16-bit_integer>`	Filters based on a 16-bit integer. Use with 16-bit processors.
`INT32 <32-bit_integer>`	Filters based on a 32-bit integer. Use with 32-bit processors.
`IOTYPE <operation type> [, <operation type>]`	Filters based on the type of operation. A list of record types can be viewed with the SHOW RECTYPE command in Logdump. Up to 32 operation types can be specified with IOTYPE.
`MATCH {ANY | ALL}`	Controls filtering response when multiple filters have been specified. Can be shortened to MAT or MA. ◆ ANY includes a record for display or counts if the condition matches any of the filter conditions. This is the default. ◆ ALL includes a record for display or counts only if the condition matches all of the filter conditions.
`OFF`	Disables record filtering. By default, filtering is disabled. An alias for this option is DISABLE.
`ON`	Enables record filtering. An alias for this option is ENABLE.
`RBA <byte address> [<comparison operator>] [...]`	Filters based on a relative byte address. Accepts either a 32-bit or 64-bit value. Up to 32 specifications can be supplied.
`RECLEN <length> [<comparison operator>]`	Filters based on a record length, in bytes. For <comparison operator>, see "Comparison operators" on page 187.

Argument	Description

`RECTYPE {<type_number> | <type_name>}`	Filters based on the type of record. Can be either of the following:
	◆ The number assigned to the record type.
	`FILTER RECTYPE 10`
	◆ The name of the record type.
	`FILTER RECTYPE Update`
	To view the record type names and numbers, issue the SHOW RECTYPE command. (See page 202.)

`SHOW`	Displays filter settings. Same as using FILTER without any options.
`STARTTIME <time_string>`	Starts the filter with the first record written at the specified time. For the time string, use the format of: [[yy]yy-mm-dd] [hh[:mm][:ss]] Example: `STARTTIME 2005-01-11 00:00:00` Can be shortened to STARTTS or START.
`STRING [BOTH] [B], <text> [<column_range>] [[B], <text> [<column_range>]] [...]`	◆ <text> filters based on a string. Enclose the string within quotes. ◆ <column_range> filters based on a range of columns. Use the format of: <start_column>:<end_column> Example: `10:35` ◆ BOTH filters on both a string and a column range. ◆ [B] specifies a case-insensitive match. You can match up to eight string and column arguments. If the trail data is EBCDIC, issue the EBCDICDATA ON or ASCIIDATA OFF command before using FILTER STRING to ensure the correct matching. STRING can be shortened to STR.
`SYSKEY <system key> [<comparison operator>] [...]`	Filters based on a NonStop source key. Accepts either a 32-bit or 64-bit value. Up to 32 specifications can be supplied.

Argument	Description
`TRANSIND <indicator> [<comparison operator>]`	Filters based on the TransInd field of the record header. Valid values: 0 = start of transaction 1 = middle of transaction 2 = end of transaction 3 = only record in transaction For example, to filter for the end of a transaction, use the following command, including the spaces in the syntax: `FILTER INCLUDE TransInd > = 2` For <comparison operator>, see "Comparison operators" on page 187.
`TRANSID '<transaction_identifier>'`	Filters on the TMF transaction identifier when reading a TMF trail, for example: `FILTER INCLUDE TRANSID \GGQA(2).0.12792182.`
`UNDOFLAG <type> [<comparison operator>]`	Filters based on the NonStop undo flag. The undo flag is set for records that are undone when a TMF transaction is aborted. Normally, UndoFlag is set to zero, but if the record is the backout of a previously successful operation, then UndoFlag will be set to 1. An undo that is performed by the disc process because of a constraint violation is not marked as an undo. For <comparison operator>, see "Comparison operators" on page 187.
`USERTOKEN <token name> [<comparison operator>] [<token value>]`	Filters based on a specific user token in the trail file header. ◆ <token name> is the name of any token that is defined with the TOKENS clause of a TABLE statement of the Extract parameter file. It is not case-sensitive. ◆ <token value> is either a constant that is enclosed within double quotes or the result of a GoldenGate column-conversion function, depending on what was specified in the TOKENS clause for <token name>. ◆ For <comparison operator>, see "Comparison operators" on page 187.

Example 1 The following shows filter options modified by comparison operators.

```
FILTER INCLUDE RECLEN > 400
FILTER INCLUDE RECLEN < 200
FILTER INCLUDE TRANSIND <> 1
FILTER INCLUDE SYSKEY > 202172700557313
```

Example 2 The following filters for a data file name and for a relative key 19446, which has a hex value of 00004bf6. Because MATCH ALL is used, a record must meet all of the filter specifications to be included in the filter.

```
FILTER INCLUDE FILENAME $QA01.QAESRC.ACCT*
FILTER INCLUDE HEX “00004bf6” 0:3
FILTER MATCH ALL
```

Example 3 The following shows filter options with multiple specifications. By default, a record that matches any of these specifications will be included in the filter. Note that in the STRING filter, two of the criteria are not case-sensitive, while one is, and the filter is confined to a column range.

```
FILTER INCLUDE IOTYPE insert,update,delete
FILTER INCLUDE STRING b”String1” “string2” b“String3” 25:50
FILTER INCLUDE FILENAME $QA01.QAESRC.ACCT1, $QA01.QAESRC.ACCT2,
$QA01.QAESRC.ACCT3
```

GHDR

Use GHDR to control whether or not the record header is displayed with each record. Each record contains a header that includes information about the transaction environment. Without arguments, GHDR displays the status of header display (ON or OFF).

Default `OFF`

Syntax `GHDR {ON | OFF}`

HEADERTOKEN

Use HEADERTOKEN to control whether or not header token indicators are displayed with each record. The header token indicators are the following:

`G` — record header (begin of record)

`H` — header area

`D` — data area

`T` — GoldenGate internal token

`U` — user token area (does not display if user tokens are not in use)

`Z` — end of record

Without arguments, HEADERTOKEN displays the status of header token indicators (ON or OFF).

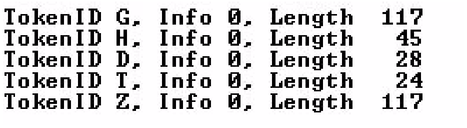
Default `OFF`

Syntax `HEADERTOKEN {ON | OFF | DETAIL}`

Argument	Description
`ON`	Enables the display of header tokens.
`OFF`	Disables the display of header tokens.

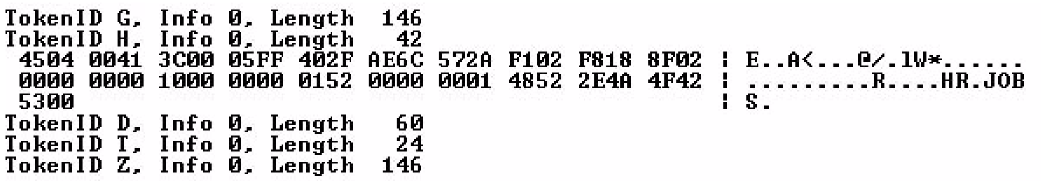
Argument	Description
`DETAIL`	Provides detailed token values.

Example 1 HEADERTOKEN, without DETAIL



```
TokenID G, Info 0, Length  117
TokenID H, Info 0, Length   45
TokenID D, Info 0, Length   28
TokenID T, Info 0, Length   24
TokenID Z, Info 0, Length  117
```

Example 2 HEADERTOKEN with DETAIL



```
TokenID G, Info 0, Length  146
TokenID H, Info 0, Length   42
 4504 0041 3C00 05FF 402F AE6C 572A F102 F818 8F02 | E..A<...@/.1W*......
 0000 0000 1000 0000 0152 0000 0001 4852 2E4A 4F42 | ..........R....HR.JOB
 5300                                              | S.
TokenID D, Info 0, Length   60
TokenID T, Info 0, Length   24
TokenID Z, Info 0, Length  146
```

HELP

Use HELP to view the syntax of Logdump commands.

Default None

Syntax `HELP`

HISTORY

Use HISTORY to view the most recently issued Logdump commands since the session started, or to reset the command count starting at line 1 again. HISTORY can be shortened to HIST.

NOTE You can use the FC command to re-execute a command in the list. See page 174.

Default Display recent commands

Syntax `HISTORY [<n>] [CLEAR]`

Argument	Description
<n>	Returns the specified number of previously issued commands, where <n> is any positive number.
`CLEAR`	Deletes the command history buffer and reverts the command line to 1.

Example `HISTORY 3`

The results of this command would be similar to:

```
1: ghdr on
2: detail on
3: scanforheader
```

INTERPRETINTERVAL

Use INTERPRETINTERVAL to display a 64-bit Julian time interval in the format of days-hh:mm:ss.ms.us.

Default None

Syntax `INTERPRETINTERVAL <interval string>`

Argument	Description
`<interval string>`	A string representing the interval to be converted.

Example `INTERPRETINTERVAL 1234567`

This produces the following result:

```
Interval 1234567 is 0-00:00:01.234.567
```

INTERPRETTIMESTAMP

Use INTERPRETTIMESTAMP to display a 64-bit Julian timestamp as an ASCII value.

Default None

Syntax `INTERPRETTIMESTAMP <timestamp>`

Argument	Description
`<timestamp>`	A JULIANTIMESTAMP value.

Example `INTERPRETTIMESTAMP 211976584185800569`

This produces the following result:

```
2005/03/03 04:29:45.800.569 GMT
2005/03/02 20:29:45.800.569 LCT
```

LOG

Use LOG to start and stop the logging of Logdump sessions. When enabled, logging remains in effect for all sessions of Logdump until disabled with the LOG STOP command. Without arguments, LOG displays the status of logging (ON or OFF). An alias for LOG is OUT.

Default Disabled

Syntax `LOG <file_name> | STOP}`

Argument	Description
`<file_name>`	Specifies the name of the log file. Specify a full path name to store the file in a directory other than the current working directory.
`STOP`	Stops logging.

Example `LOG $data01.glogggl.sesslog`

NEXT

Use NEXT to display the next record or records in the file. The default displays only the next record. NEXT can be shortened to N. An alias for NEXT is RECORD.

Default Display the next 1 record

Syntax `NEXT [<n>]`

Argument	Description
`<n>`	Displays the specified number of subsequent records.

Example `NEXT 10`

NEXTTRAIL

Use NEXTTRAIL to close an open trail file and open the next one in the sequence. An alias for NEXTTRAIL is NT.

Default None

Syntax `NEXTTRAIL`

NOTIFY

Use NOTIFY to display the number of records scanned, the trail position, and the record timestamp at specified intervals when using COUNT and records are being suppressed from display through filtering options. An alias for NOTIFY is NOTIFYINTERVAL.

Instead of displaying each notify interval on a separate line, you can configure Logdump to simply update a single line with each new scan result. See “SCANSCROLLING” on page 202.

Default None

Syntax `NOTIFY <interval>`

Argument	Description
`<interval>`	The notification interval expressed as a number of records.

Example The following shows the usage and result of this command.

```
Logdump 26> NOTIFY 1000
Logdump 27> FILTER INCLUDE FILE sales.res*
Logdump 28> COUNT
Scanned 1000 records, RBA 160380,2005/02/24 08:53:47.768.255
Scanned 2000 records, RBA 729961,2005/02/24 08:56:09.916.128
Scanned 3000 records, RBA 2032683,2005/02/24 08:56:09.916.128
Scanned 4000 records, RBA 3244585,2005/02/24 08:56:09.916.128
Scanned 5000 records, RBA 4568766,2005/02/24 08:56:09.916.128
```

OBEY

Use OBEY to process a file that contains a list of Logdump commands. OBEY is useful for executing commands that are frequently used in sequence.

OBEY can be shortened to O. An alias for OBEY is SOURCE.

Default None

Syntax `OBEY <file name>`

Argument	Description
`<file name>`	The fully qualified name of the file containing the list of commands.

Example OBEY $DATA01.GGSPARM.OBEY1

The preceding command executes a file that might look something like this:

```
ghdr on
usertoken on
detail
filter enable
filter clear
filter match all
```

OPEN

Use OPEN to open a trail file or extract file in Logdump. Without arguments, the command displays the name of the file that is currently open. Aliases for OPEN are FROM and LOGTRAIL.

Default None

Syntax `OPEN <file_name>`

Argument	Description
`<file_name>`	The fully qualified path name of the trail file or extract file to be opened. To specify a trail file, specify the trail name (a two-character prefix) and the sequence number, for example jd000000.

Example OPEN $data01.glogggl.aa000000

POSITION

Use POSITION to set the read position in the file. The position of a record in the file is noted in the record header in the AuditPos field.

Without options, POSITION displays the current read position. Options let you specify an exact position. After you set the position, issue the NEXT command to view the record at that position.

POSITION can be shortened to POS.

Default None

Syntax `POSITION [<bytes> | {0 | FIRST}]`

Argument	Description
`<bytes>`	Specifies the number of bytes into the file at which to read. Use the NEXT command to view the specified record.
`0 | FIRST`	Positions Logdump at the beginning of the file.

Example `POS 77580548`

RECLEN

Use RECLEN to control how much of the record data is displayed. You can use RECLEN to control the amount of scrolling that must be done when records are large, while still showing enough data to evaluate the record. Data beyond the specified length is truncated.

Default 140 bytes

Syntax `RECLEN <n>`

Argument	Description
`<n>`	The number of bytes of the record that is displayed.

Example `RECLEN 280`

SAVE

Use SAVE to write a subset of the records to a new trail or extract file. By saving a subset to a new file, you can work with a smaller file that is easier to debug. Saving to another file also enables you to extract valid records that can be processed by GoldenGate, while excluding records that may be causing errors.

To set the version of the trail or file (to old or new format), use the TRAILFORMAT command.

Default None

Syntax

```
SAVE <file_name> [!] {<n> records | <n> bytes}
[EXT ( <pri>, <sec> [, <max>])]
[MEGABYTES <n>]
[NOCOMMENT]
[OLDFORMAT | NEWFORMAT]
[TRANSIND <indicator>]
[TRUNCATE]
```

Argument	Description
`<file_name>`	The name of the new file. To specify a trail file, specify the two-character trail name and a sequence number, for example rt000001.
`!`	Overwrites the specified file, if the same file already exists. First a purge is done, and then the specified records are saved to the file.

`<n> records |`
`<n> bytes`
	Specifies either a number of records or a number of data bytes to write to the new file. The <n> number of records or bytes are taken forward from the current position in the file. You can change the position with the POSITION command. See page 198.

`EXT ( <pri>, <sec>`
`[, <max>])`
	Specifies savefile extent sizes.

`MEGABYTES <n>`	Specifies the size of a savefile extent.
`NOCOMMENT`	Suppresses the leading and trailing comment records that are placed by default in the new file. These records describe the context of the file. The begin comment record contains source trail information and the position where the save started. The end comment record identifies the end of the saved data. These headers are useful to separate different sets of records that are saved to the same file, but can be omitted.

`OLDFORMAT |`
`NEWFORMAT`
	Writes the data in either the current trail format (NEWFORMAT, the default) or the format that was used for GoldenGate versions 6.0 and earlier (OLDFORMAT).

Argument	Description

`TRANSIND <indicator>`	Sets the TransInd header field in the records written to one of the following:
	`FIRST`
	`MIDDLE`
	`END`
	`ONLY`
	This allows you to reorder records in a transaction. TRANSIND applies to all records written by a SAVE command.

`TRUNCATE`	Purges an existing file before saving new information to it.

Example `SAVE $data01.glogggl.ss000000 100 records`

SCANFORENDTRANSACTION

Use SCANFORENDTRANSACTION to scan for a record that has a transaction indicator of 2 or 3, as shown in the TransInd field of the header. When one of those indicators is found, Logdump displays the first record of the next transaction.

The indicators represent the following:

- 2 — last record in the transaction
- 3 — only record in the transaction

SCANFORENDTRANSACTION can be shortened to SFET.

Default None

Syntax `SCANFORENDTRANSACTION`

SCANFORHEADER

Use SCANFORHEADER to go to the next record header. Before using this command, use the GHDR ON command to show record headers (see page 193). SCANFORHEADER can be shortened to SFH.

Default None

Syntax `SCANFORHEADER [PREV]`

Argument	Description
PREV	Displays the previous record header.

SCANFORRBA

Use SCANFORRBA to scan for the record at a relative byte address specified by the AuditRBA field of the record header. Before using this command, use the GHDR command to show record headers (see page 193). SCANFORRBA can be shortened to SFR.

Default None

Syntax `SCANFORRBA <relative byte address> [<file_name>]`

Argument	Description
`<relative byte address>`	Specifies the relative byte address to find.
`<file_name>`	Constrains the search to an Enscribe or SQL data file. A file name is required even if you are searching a file that is open in Logdump.

Example SCANFORRBA 321 $data01.glogggl.rt000000

SCANFORTIME

Use SCANFORTIME to scan for a record that contains a specific timestamp. The timestamp is contained in the IO Time field of the record header. Before using this command, use the GHDR command to show record headers (see page 193). SCANFORTIME can be shortened to SFTS.

Default None

Syntax `SCANFORTIME <time_string> [, <name>]`

Argument	Description
`<time_string>`	Scans for a specific timestamp. For the time string, use the format of: [[yy]yy-mm-dd] [hh[:mm][:ss]]
`<name>`	Constrains the search to a specific table or data file name, or a group of names specified with a wildcard.

Example SCANFORTIME `2005-10-27 14:33:57`

SCANFORTYPE

Use SCANFORTYPE to scan for the next record of the specified type. SCANFORTYPE can be shortened to SFT.

Default None

Syntax `SCANFORTYPE {<type_name> | <type_number>}`

Argument	Description
`<type_name> | <type_number>`	Specifies the type of record to search for, either by type name or type number. To view a list of record types and their associated numbers, use the SHOW RECTYPE command (see page 202).

Example Both of the following commands return the same result: They display commit records.

```
SCANFORTYPE Commit
SFT 2
```

SCANSCROLLING

Use SCANSCROLLING to configure Logdump to update a single line after COUNT scans when NOTIFY is enabled. Otherwise, each scan notification appears on a different line. See "NOTIFY" on page 196 for more information.

Default `OFF`

Syntax `SCANSCROLLING {ON | OFF}`

Argument	Description
`ON`	Enables the use of a single line for count notification results.
`OFF`	Disables the use of a single line, causing a separate line to be used for each notification.

SHOW

Use SHOW to display internal Logdump information, including files that are open, the current Logdump environment, a list of GoldenGate record types, and current filter settings. SHOW can be shortened to SH or SHO.

Default None

Syntax

```
SHOW
[ENV]
[FILTER]
[OPEN]
[RECTYPE]
```

Argument	Description
`ENV`	Displays the current Logdump environment. Same as the ENV command (see page 173).
`FILTER`	Displays current filter settings.

Argument	Description
`OPEN`	Shows all NonStop files that are open in Logdump.
`RECTYPE`	Displays a list of GoldenGate record types that can be displayed with Logdump.

Example 1 `SHOW FILTER`

This shows something similar to the following:

```
Data filters are ENABLED
Include  Match ALL
Filename-0 : $QA01.QAESRC.ACCT*
HEX-0      : ( 4), Col 0:3
0000 4BF6
Exclude  Match ANY
```

Example 2 `SHOW OPEN`

This shows something similar to the following:

```
0 : $RECEIVE
1 : \GGS2.$ZTN2.#PTW6EUX
2 : \GGS2.$DATA4.#0009047
3 : \GGS2.$ZTN2.#PTW6EUX
4 : \GGS2.$DATA4.CPSDAT.TM000000
```

Example 3 `SHOW RECTYPE`

This shows results similar to the following. (This list might not reflect all possible record types. New types are added when needed to support new functionality.)

```
LogTrail record types
   1 - Abort                             2 - Commit
   3 - Delete                            4 - EndRollBack
   5 - Insert                            6 - Prepared
   7 - TMF-Shutdown                      8 - TransBegin
   9 - TransRelease                     10 - Update
  11 - UpdateComp                       12 - FileAlter
  13 - FileCreate                       14 - FilePurge
  15 - FieldComp                        16 - FileRename
  17 - AuxPointer                       18 - NetworkCommit
  19 - NetworkAbort                     20 - CurrentPos
  89 - SQL/MX DDL OP                    90 - GGSSQLCol
 100 - GGSPurgedata                    101 - GGSPurgeFile
 102 - GGSCreateFile                   103 - GGSAlterFile
 104 - GGSRenameFile                   105 - GGSSetmode
 107 - GGSControl                      106 - GGSChangeLabel
 160 - DDL OP                         115 - GGSKeyFieldComp
 117 - GGSKeyFieldComp32               161 - RecordFragment
 116 - LargeObject                  132 - GGSCreateSequence
 133 - GGSAlterSequence               134 - GGSDropSequence
 150 - RestartAbend                    151 - RestartOK
 152 - RecoveryEnd                     200 - GGSBulkio
 201 - GGSFileClose                    202 - GGSLoggerTS
 203 - GGSExtractTS                    204 - GGSCollectTS
 205 - GGSComment                      250 - LibOpenTrace
 251 - LibCloseTrace                  252 - LoggerOpenTrace
 253 - LoggerCloseTrace               254 - LoggerAddedInfo
 249 - LoggerAddedStats
```

SKIP

Use SKIP to skip the specified number of records.

Default None

Syntax `SKIP <n>`

Argument	Description
<n>	The number of records to skip.

Example `SKIP 50`

TIME

Use TIME to display the current time in local and GMT formats.

Default None

Syntax `TIME`

TIMEOFFSET

Use TIMEOFFSET to set the Logdump time format. Without arguments, TIMEOFFSET displays the current time offset. Options enable you to set the time to the local time, Greenwich Mean Time (GMT), or a specific offset from GMT. The specified time format applies to the timestamps shown in records as well as any Logdump commands that accept a time string argument.

Default `LOCAL`

Syntax `TIMEOFFSET {LOCAL | GMT | GMT + <hh[:mm]> | GMT - <hh[:mm]>}`

Argument	Description
`LOCAL`	Sets the time to that of the local system.
`GMT`	Sets the time to Greenwich Mean Time (GMT).
`GMT + <hh[:mm]>`	Sets the time ahead of GMT by the specified number of hours and, optionally, minutes.
`GMT - <hh[:mm]>`	Sets the time behind GMT by the specified number of hours and, optionally, minutes.

Example `TIMEOFFSET GMT -01`

TMFBEFOREIMAGE

Use TMFBEFOREIMAGE to view the before image for update operations from TMF audit.

Default `OFF`

Syntax `TMFBEFOREIMAGE {ON | OFF}`

Argument	Description
`ON`	Displays the before image for update operations from the TMF audit.
`OFF`	Displays only the after image for update operations from the TMF audit.

Example A sample display for TMFBEFOREIMAGE ON is shown below.

```
2006/12/12 10:02:34.325.264 FieldComp          Len    38 RBA 615854956
Name: \NY.$DATA1.GGSDAT.TCUSTMER
Before Image:                                           Partition 0
0000 0004 414E 4E20 0002 0014 5345 4154 544C 4520 | ....ANN ....SEATTLE
2020 2020 2020 2020 2020 2020 0003 0002 5741      |             ....WA
2006/12/12 10:02:34.325.264 FieldComp          Len    38 RBA 615854956
Name: \NY.$DATA1.GGSdat.TCUSTMER
After  Image:                                           Partition 0
TRANSID     : \NY(2).0.7022034  (7998393398406021122)
0000 0004 414E 4E20 0002 0014 4E45 5720 594F 524B | ....ANN ....NEW YORK
2020 2020 2020 2020 2020 2020 0003 0002 4E59      |             ....NY
```

TRAILFORMAT

Use TRAILFORMAT to set the version of the GoldenGate trail or extract file that is being saved when using the SAVE command.

Default NEW

Syntax `TRAILFORMAT {NEW | OLD}`

Argument	Description
NEW	Sets the format to that used by GoldenGate version 6.0 and later.
OLD	Sets the format to that used by GoldenGate versions earlier than 6.0.

TRANSBYTELIMIT

Use TRANSBYTELIMIT to prevent normal-sized transactions from being tracked in the transaction table specified with the TRANSHIST command. It sets a lower boundary for the number of bytes in a transaction and should be set to represent a normal-sized transaction for the environment being evaluated with Logdump. Setting a boundary reduces the amount of data that is stored and, consequently, the amount that must be reviewed when troubleshooting.

Default 10000 bytes

Syntax `TRANSBYTELIMIT <n>`

Argument	Description
<n>	The number of bytes in a normal-sized transaction.

Example `TRANSBYTELIMIT 9000`

TRANSHIST

Use TRANSHIST to keep track of the size of transactions in a trail or file. Logdump tracks the transactions in an internal history table in descending order according to the number of bytes of data in each one. When the history table is full, the smallest transaction is removed to allow a larger transaction to be added to the list.

Use TRANSHIST in conjunction with other Logdump commands to determine whether or not your applications generate large transactions and to identify their relative size. This information can be used when deciding how to group tables into different processing groups for faster throughput. For more information, see page 163.

NOTE You can use the SEND EXTRACT command with the SHOWTRANS option to view a list of long-running transactions. Other options enable you to control whether those transactions are ignored or processed by GoldenGate.

Default 0 (do not maintain history)

Syntax `TRANSHIST <n>`

Argument	Description
`<n>`	Sets the size of the history table, in bytes. Valid values are 0 through 200 bytes. A value of 0 means that no transaction history is maintained.

Example `TRANSHIST 150`

TRANSRECLIMIT

Use TRANSRECLIMIT to prevent normal-sized transactions from being tracked in the transaction table specified with the TRANSHIST command. It sets a lower boundary for the number of records in a transaction and should be set to represent a normal-sized transaction for the environment being evaluated with Logdump. Setting a boundary reduces the amount of data that is stored and, consequently, the amount that must be reviewed when troubleshooting.

Default 100 operations

Syntax `TRANSRECLIMIT <n>`

Argument	Description
`<n>`	The number of records in a normal-sized transaction.

Example `TRANSRECLIMIT 90`

USERTOKEN

Use USERTOKEN to control whether or not user token data is displayed with each record. A user token is data specified by a GoldenGate user that is stored in the record header and can be mapped to a target column or used for some other purpose during GoldenGate processing.

Without arguments, USERTOKEN displays the status of user token display (ON or OFF). With the ON option, the name of the token and its length are displayed. The DETAIL option shows the actual token data.

Figure 35 User tokens, with data

```
User tokens        7 bytes
5465 7374 0031 00                  | Test.1
```

Default Display token name and length.

Syntax `USERTOKEN {ON | OFF | DETAIL}`

Argument	Description
`ON`	Enables the display of user tokens.
`OFF`	Disables the display of user tokens.
`DETAIL`	Displays the token data.

VOLUME

Use VOLUME to set the default volume or subvolume. An alias for this command is CD.

Default None

Syntax `VOLUME <volume, or subvolume>`

WRITELOG

Use WRITELOG to write text to the session log. Before using this command, start logging with the LOG command (see page 195).

Default None

Syntax `WRITELOG <text>`

Argument	Description
`<text>`	Any text string. Quotes are optional.

Example `WRITELOG “Customer name is ABC Company.”`

X

Use X to execute a program from within Logdump. When you exit the program, the Logdump prompt returns.

Default None

Syntax `X <program> [<string>]`

Argument	Description
`<command>`	The program to run.
`<string>`	A character string, such as input arguments.

Example The following series of commands and output shows how you can exit Logdump, issue other commands from the shell or within GGSCI, and then return to the Logdump command line.

```
Logdump 696 >x ggsci

GoldenGate Command Interpreter
Version .....

GGSCI (sysa) 1> status er *
GGSCI (sysa) 2> start er *
GGSCI (sysa) 3> info er *
GGSCI (sysa) 4> exit
Logdump 697 >
```

APPENDIX 1
GoldenGate Components

The following tables outline components of GoldenGate code used for customer implementations.

Programs, utilities, macros, and libraries

Program/ Macro	Description
AUDDUMP	A utility program to print the contents of the AUDCFG or GGSCPUxx segments.
AUDSERV	Reads audited database changes from TMF audit trails. Started by Extract processes. Must be owned by SUPER group and must have a PROG ID and license for other users to extract changed data from the audit trails.
AUDSERVN	Native version of Audserv.
BASELIB	The intercept library that is bound with programs to facilitate non-TMF based database change logging. The TNS mode can be used with programs written in TAL, and C.
BASELIBN	Native version of BASELIB that can be used with TAL, C, C++, and JavaVM. On H06/J06 operating systems, this library can be also be used with native COBOL programs.
BINDEXIT	TACL macro that merges Extract and Replicat with user exit routines.
BINDSKEL	TACL macro that merges BASELIB with the BASE24 SKELB library.
CHGNOTE	Notifies GGSLIB of configuration changes.
COMBLIB	A macro that combines the GoldenGate intercept library GGSLIB with a User Library that intercepts the same calls as GoldenGate.
COORDINATOR	Tracks the status of distributed network transactions to coordinate the processing across multiple nodes.
DBINIT	A macro that initializes the GoldenGate checkpoint and configuration files.

Program/ Macro	Description
DDLGEN	Generates target database table definitions based on NonStop DDL.
DEFGEN	Generates source table definitions for Replicat to use for the translation of heterogeneous databases.
DIFFFLS	A utility that compares two Enscribe data files and reports on any out-of-sync conditions.
EMSDIST	Contains EMS messaging and tokens for GoldenGate messages.
EMSINST	A macro that installs EMS event message detail text.
ENTRYLIB	A macro that combines the GoldenGate intercept library BASELIBE with a User Library that intercepts the same calls as GGSLIB.
EVCXFUP	Contains the FUP scripts to create the EVENTCX file for custom EMS messages.
EXTRACT	Performs database change extract; formats and outputs results.
EXTRACTN	Native version of Extract.
EXTRR	Native relinkable version of Extract. Use when you have a native user exit to be linked into Extract. The NLDEXIT macro will reference this program.
FIXFLS	Adjusts alternate key pointers and turns auditing on proper GoldenGate files (for example, after duplicating from another location).
FUPLOG	Contains the FUP scripts to create the Logger configuration file LOGCONF.
GGSCI	User interface to GoldenGate functions.
GGSDLL	The default name of the executable version of native BASELIB for applications running on H06/J06 operating systems. NLDLIB may combine BASELIB with other relinkables when creating GGDLL.
GGSLIB	User library bound with application programs to facilitate non-TMF based database change extract.
GGSLIBR	Native relinkable version of GGSLIB that can be combined with other code to create a larger combined user library.
GGSSRL	The default name of the executable version of native BASELIB for applications running on D46 and greater or G06 operating systems. NLDLIB may combine BASELIB with other relinkables when creating GGSSRL.
INSTALL	TACL macro that creates database files and performs other installation tasks.
KEYGEN	Program used to generate random keys. Used for encryption.

Program/ Macro	Description
LEANBIND	A macro that removes the several routines out of BASELIB and creates LEANLIB. Use in place of BASELIB; however, you must contact Technical Support at GoldenGate before proceeding.
LOGDUMP	A program that provides the ability to display or search for information stored in log trails, extract trails, or extract flat files.
LOGDUMPT	A version of LOGDUMP specific to TMF-audit trails.
LOGGER	Writes non-TMF audited database changes to log trails.
MEASFLS	A utility that interfaces with Measure to collect data about file I/O at specified intervals.
MEASRPT	A utility that generates reports based on data collected from MEASFLS.
MGR	Carries out resource management functions as configured by system administrators.
MIGRATE	Moves checkpoint and other data from an old to a new GoldenGate installation.
NLDEXIT	A macro used to combine a native version of the intercept libraries.
NLDLIB	A macro used to create a native version of the intercept libraries.
READER	Monitors trails for distributed network transactions and communicates status information to the Coordinator.
REPLICAT	Replicates selected database changes from a set of source Enscribe files and SQL tables to a set of target files and tables.
REPNP	The TNS version of Replicat; does not contain the privilege code procedures. This can be used for debugging user exits without requiring SUPER access.
REPR	A relinkable, native version of Replicat. Used when linking native user exits.
REVERSE	"Reverses" operations in an extract file so that Replicat can selectively back out changes rather than replicate them.
SCANGRP	A utility used by the MIGRATE utility to scan and convert group records.
SEGDUMP	A utility program used to examine the contents of the private context segment maintained by BASELIB. The context segment contains I/O buffers and state information for all files being logged.
SERVER	The Extract server collector that receives data over TCP/IP and writes data to remote trails.

Program/ Macro	Description
SFGEXIT	A program that can be activated within NonStop's Safeguard to audit update access file opens and issue a message if no GoldenGate intercept program is bound in.
SQLCOMPS	List of SQLCOMP commands for GoldenGate programs.
SYNCFILE	Performs file replication of non-database files based on a user-set schedule.
SYNCTAB	A utility that compares two NS SQL/MP tables and reports on any out-of-sync conditions.
TMFARLB2	A distributed TMF audit read library.
TMFARUL2	A distributed TMF audit read library.
UNPAKIT	A macro that executes the UNPAK against GoldenGate installation files and runs the installation macro.

GoldenGate database

The GoldenGate database is created at installation time with the INSTALL macro. Each database file and its function are listed below.

Database File	Description
AUDCFG	A dynamically created shared segment file used for GGSLIB configuration retrieval.
AUDSPEC	Contains audit management configuration parameters set up with GGSCI and read by Manager to carry out audit management tasks.
CONTEXT	Contains Extract checkpoints to facilitate continuous processing of audit for particular Extract groups. Used by Manager to determine whether or not particular audit resources are still required.
EXTCTXT	Contains Extract checkpoints to track restart points within extract files in case Extract halts prematurely. Also contains information about individual extract trail dimensions and management.
EXTCTXT0	Alternate key file for EXTCTXT.
GROUP	Contains each distinct Extract processing group.
LOGCONF	Stores logger configuration.
LOGGGS	Keeps log of critical events.

Database File	Description
`MRKRGGS`	Audited file that accepts audit marker records.
`REMCTXT`	Contains the names of local and remote REPCTXT files to check before deleting extract files.
`REPCTXT`	Contains Replicat checkpoints to facilitate continuous processing of extract information. Used by Manager to determine whether or not particular extract file resources are still required.
`REPCTXT0`	Alternate key file for REPCTXT.
`REPGRP`	Contains each distinct Replicat processing group.
`RMTCTXT`	Contains checkpoints for remote extract files.
`SYNCGRP`	Contains each distinct Syncfile processing group.

External component summary

Component	Description
TMF Audit Trails	Contain change data information for source file and table insert, update and delete operations. Audit trails are read by Extract processes with Audserv for relevant database changes. Each audit trail can be read from one of four locations: the original location (on disk), from a disk dump, restored to disk from tape, or from a duplicate created by Manager.
Extract Files	Created by Extract processes. Extract files contain formatted database change records that can be input to Replicat processes, user-written applications or utilities. Extract processes can write to a single extract file, or a sequence of extract files known as *extract trails*. Manager purges extract files.
SQL and Enscribe Database Files	Database files that are the source of extract and target of delivery activities.
Parameter Files	Parameter files provide run-time parameters for all GoldenGate processes. There is no facility for entering parameters interactively, so parameter files are required.
Report Files	Extract and Replicat generate reports detailing statistical highlights of processing. Report files can be virtually any format, including spooler, edit files or the home terminal.

Component	Description
DDL Dictionaries	Dictionaries are an optional component of Extract and Replicat processing. DDL dictionaries describe Enscribe records processed by Extract and Replicat. This lets you describe record selection and column mapping criteria for both Extract and Replicat activities. The dictionaries must have been compiled using C30 or later DDL.
SQL Catalogs	Provide the definition of extracted SQL database change records. You need read access to the catalog associated with any source and target SQL tables. The catalogs allow Extract and Replicat to decode audit records and to build replicate SQL operations.
User Exit Routines	Object files you create that are bound in with Extract or Replicat to perform customized routines.
Error Console	Extract and Replicat send errors and warnings to both the error console and to the report file. The default error console is the home terminal. You can change the error console with the statement PARAMS EMSLOG <EMS collector> before issuing the run command for either program. <file name> can include EMS distributors, processes, disk or spooler files.

Templates, demonstrations, and sample code

File	Description
`DDLEXIT`	DDL for creating copy files for user exit and API routines.
`DEMOCL`	C include library for user exit demos.
`DEMOCOBL`	COBOL copy library for user exit demos.
`DEMOEDDL`	Sample DDL scripts for sample Enscribe files.
`DEMOFUPS`	Sample FUP scripts to create sample Enscribe source files.
`DEMOFUPT`	Sample FUP scripts to create sample Enscribe target files.
`DEMOOLDE`	Source code for the DEMOLEDO Enscribe demo program that executes inserts and updates for sample files.
`DEMOLDEO`	Program used to generate insert and update operations for sample Enscribe files.
`DEMOLDS`	Sample data for tables created by DEMOSQL.

File	Description
DEMORCOB	An example program for reading extract trails using the API.
DEMOSQL	SQL table creation DDL for sample TMF delivery User Tutorial example.
DEMOXC	An example user exit written in C.
DEMOXCOB	An example user exit written in COBOL.
HELP	Help file for GGSCI.
TCPERRS	Contains TCP/IP error handling parameters.
TMPLMSA	DDLGEN template file for MS Access.
TMPLMSS	DDLGEN template file for MS SQL Server.
TMPLORA	DDLGEN template file for Oracle.
TMPLTDM	DDLGEN template file for Non Stop NS SQL.
TMPLSYB	DDLGEN template file for Sybase.
TMPLDB2	DDLGEN template file for DB2 on Windows and UNIX.
TMPLDB2M	DDLGEN template file for DB2 on z/OS and OS/390.
USEREXC	Blank template that can be used as a starting point for C user exits.
USEREXT	Blank template that can be used as a starting point for TAL user exits.
XLIBC	Include library for writing C user exits and interfacing to API.
XLIBCOB	Copy library for writing COBOL user exits and interfacing to API.

APPENDIX 2

Installing Event Detail Text

Event Text makes GoldenGate detailed text available to operators using the Viewpoint event detail screen. This appendix provides instructions for installing and working with Event Text.

Standard installation

To install event text (assuming a default configuration of Viewpoint), issue the following commands.

```
TACL> VOLUME <GoldenGate installation volume and subvolume>
TACL> RUN EMSINST
```

This macro executes the steps described in the following section.

Custom installation

If your installation of Viewpoint differs from the default configuration, perform the following steps (substitute appropriately for PATHMON process names, and so on).

1. From TACL, enter the following:

   ```
   TACL> VOLUME GGS
   TACL> FUP /IN EVCXFUP/
   TACL> RUN EVCXLDO /IN EMSDTL/
   ```

2. Bring down the Viewpoint event detail server, as shown in these commands:

   ```
   TACL> PATHCOM $ZVPT
   =FREEZE SERVER ZVPT-EVNT-DETL
   =STOP SERVER ZVPT-EVNT-DETL
   =STOP SERVER ZVPT-EVNT-DETL
   ```

   If this is the first time that custom messages will be used on your system also perform the following assignment.

   ```
   =ALTER SERVER ZVPT-EVNT-DETL, ASSIGN CUSTOM-DETAIL,
   \SYS.$SYSTEM.SYSTEM.EVENTCX

   =EXIT
   ```

3. If $SYSTEM.SYSTEM.EVENTCX already exists, enter:

   ```
   TACL> FUP COPY EVENTCX, $SYSTEM.SYSTEM.EVENTCX, SHARE
   ```

   Otherwise, enter:

   ```
   > FUP DUP EVENTCX, $SYSTEM.SYSTEM.EVENTCX
   ```

4. Bring up the Viewpoint event detail server, as follows:

   ```
   TACL> PATHCOM $ZVPT
   =THAW SERVER ZVPT-EVNT-DETL
   =START SERVER ZVPT-EVNT-DETL
   =EXIT
   ```

Customizing error messages

You can customize message text by altering the event text within the EMSDTL file before running the EVCXLDO or EMSINST programs.

APPENDIX 3

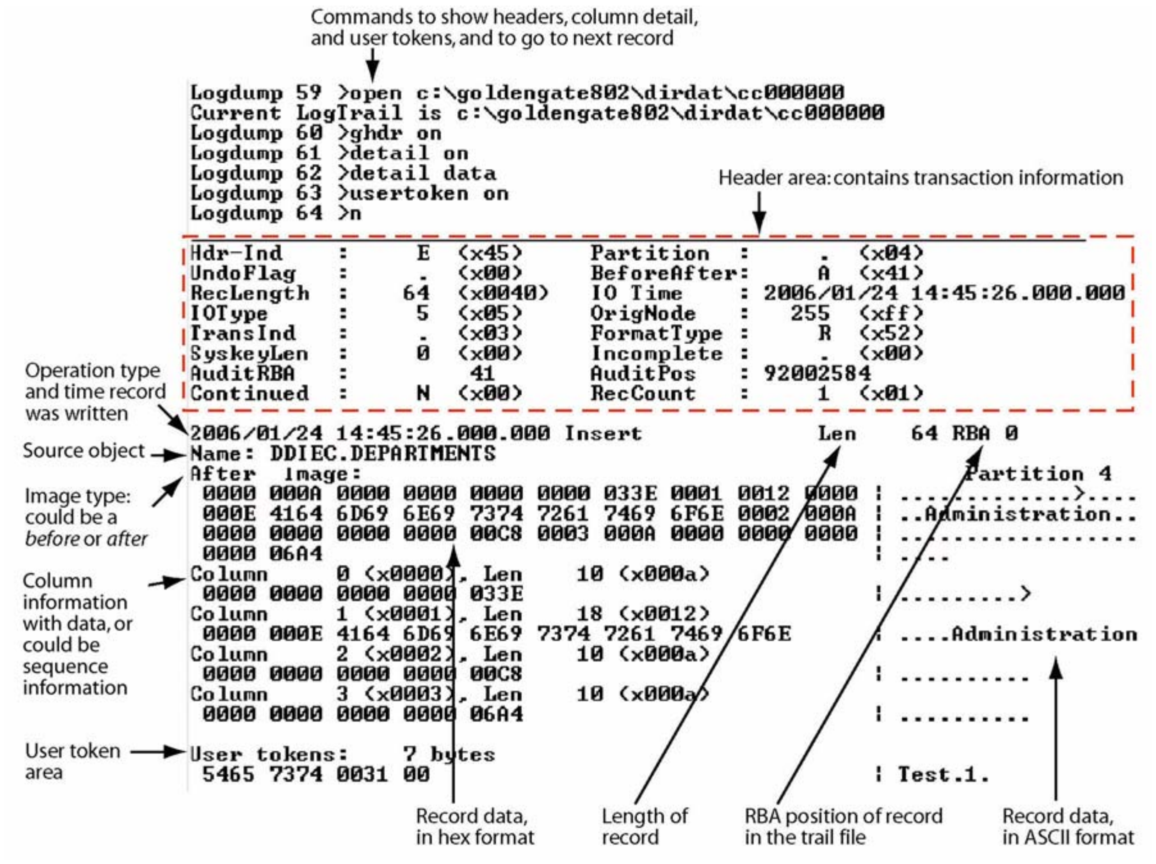
GoldenGate Record Format

This appendix describes the format of GoldenGate records that are written to a trail or extract file. Each change record written by GoldenGate to a trail or extract file includes a header area (unless the NOHEADERS parameter was specified), a data area, and possibly a user token area. GoldenGate trail files are unstructured. You can view GoldenGate records with the Logdump utility provided with the GoldenGate software. For more information, see the Logdump documentation in the *Troubleshooting and Tuning Guide*.

Example of a GoldenGate record

The following illustrates a GoldenGate record as viewed with Logdump. The first portion (the list of fields) is the header and the second portion is the data area. The record looks similar to this on all platforms supported by GoldenGate.

Figure 36 Sample trail record as viewed with the Logdump utility


Commands to show headers, column detail, and user tokens, and to go to next record
Header area: contains transaction information
Operation type and time record was written
Source object
Image type: could be a *before* or *after*
Column information with data, or could be sequence information
User token area
Record data, in hex format
Length of record
RBA position of record in the trail file
Record data, in ASCII format

```
Logdump 59 >open c:\goldengate802\dirdat\cc000000
Current LogTrail is c:\goldengate802\dirdat\cc000000
Logdump 60 >ghdr on
Logdump 61 >detail on
Logdump 62 >detail data
Logdump 63 >usertoken on
Logdump 64 >n

Hdr-Ind    :     E  (x45)     Partition   :     .  (x04)
UndoFlag   :     .  (x00)     BeforeAfter:      A  (x41)
RecLength  :    64  (x0040)   IO Time     : 2006/01/24 14:45:26.000.000
IOType     :     5  (x05)     OrigNode    :   255  (xff)
TransInd   :     .  (x03)     FormatType  :     R  (x52)
SyskeyLen  :     0  (x00)     Incomplete  :     .  (x00)
AuditRBA   :        41        AuditPos    : 92002584
Continued  :     N  (x00)     RecCount    :     1  (x01)

2006/01/24 14:45:26.000.000 Insert                 Len    64 RBA 0
Name: DDIEC.DEPARTMENTS
After  Image:                                              Partition 4
 0000 000A 0000 0000 0000 0000 033E 0001 0012 0000 | ..............>....
 000E 4164 6D69 6E69 7374 7261 7469 6F6E 0002 000A | ..Administration..
 0000 0000 0000 0000 00C8 0003 000A 0000 0000 0000 | ..................
 0000 06A4                                         | ....
Column     0 (x0000), Len    10 (x000a)
 0000 0000 0000 0000 033E                          | .........>
Column     1 (x0001), Len    18 (x0012)
 0000 000E 4164 6D69 6E69 7374 7261 7469 6F6E      | ....Administration
Column     2 (x0002), Len    10 (x000a)
 0000 0000 0000 0000 00C8                          | ..........
Column     3 (x0003), Len    10 (x000a)
 0000 0000 0000 0000 06A4                          | ..........

User tokens:    7 bytes
 5465 7374 0031 00                                 | Test.1.
```

Record header area

The GoldenGate record header provides metadata of the data that is contained in the record and includes the following information.

- The operation type, such as an insert, update, or delete
- The before or after indicator for updates
- Transaction information, such as the transaction group and commit timestamp

Description of header fields

The following describes the fields of the GoldenGate record header. Some fields apply only to certain platforms.

Table 16 GoldenGate record header fields

Field	Description
Hdr-Ind	Should always be a value of E, indicating that the record was created by the Extract process. Any other value indicates invalid data.
UndoFlag	(NonStop) Conditionally set if GoldenGate is extracting aborted transactions from the TMF audit trail. Normally, UndoFlag is set to zero, but if the record is the backout of a previously successful operation, then UndoFlag will be set to 1. An undo that is performed by the disc process because of a constraint violation is not marked as an undo.
RecLength	The length, in bytes, of the record buffer.
IOType	The type of operation represented by the record. See Table 17 on page 225 for a list of operation types.
TransInD	The place of the record within the current transaction. Values are: 0 — first record in transaction 1 — neither first nor last record in transaction 2 — last record in the transaction 3 — only record in the transaction
SyskeyLen	(NonStop) The length of the system key (4 or 8 bytes) if the source is a NonStop file and has a system key. If a system key exists, the first Syskeylen bytes of the record are the system key. Otherwise, SyskeyLen is 0.
AuditRBA	The relative byte address of the commit record. All records in a transaction will have the same commit relative byte address. The combination of IO Time and AuditRBA uniquely identifies data from a given transaction.

Table 16 GoldenGate record header fields (continued)

Field	Description
Continued	(Windows and UNIX) Identifies whether or not the record is a segment of a larger piece of data that is too large to fit within one record. LOBs, CLOBS, and some VARCHARs are stored in segments. Y — the record is a segment; indicates to GoldenGate that this data continues to another record. N — there is no continuation of data to another segment; could be the last in a series or a record that is not a segment of larger data.
Partition	This field is for GoldenGate internal use and may not be meaningful for any particular database. For Windows and UNIX records, this field will always be a value of 4 (FieldComp compressed record in internal format). For these platforms, the term "Partition" *does not* indicate that the data represents any particular logical or physical partition within the database structure. For NonStop records, the value of this field depends on the record type: ◆ In the case of BulkIO operations, Partition indicates the number of the source partition on which the bulk operation was performed. It tells GoldenGate which source partition the data was originally written to. Replicat uses the Partition field to determine the name of the target partition. The file name in the record header will always be the name of the primary partition. Valid values for BulkIO records are 0 through 15. ◆ For other non-bulk NonStop operations, the value can be either 0 or 4. A value of 4 indicates that the data is in FieldComp record format.
BeforeAfter	Identifies whether the record is a before (B) or after (A) image of an update operation. Inserts are always after images, deletes are always before images.
IO Time	The timestamp of the commit record, in local time of the source system, in GMT format. All records in a transaction will have the same commit timestamp. The combination of IO Time and AuditRBA uniquely identifies data from a given transaction.
OrigNode	(NonStop) The node number of the system where the data was extracted. Each system in a NonStop cluster has a unique node number. Node numbers can range from 0 through 255. For records other than NonStop in origin, OrigNode is 0.

Table 16 GoldenGate record header fields (continued)

Field	Description
FormatType	Identifies whether the data was read from the transaction log or fetched from the database. F — fetched from database R — readable in transaction log
Incomplete	This field is obsolete.
AuditPos	Identifies the position of the Extract process in the transaction log.
RecCount	(Windows and UNIX) Used for LOB data when it must be split into chunks to be written to the GoldenGate file. RecCount is used to reassemble the chunks.

Using header data

Some of the data available in the GoldenGate record header can be used for mapping by using the GGHEADER option of the @GETENV function or by using any of the following transaction elements as the source expression in a COLMAP statement in the TABLE or MAP parameter.

- GGS_TRANS_TIMESTAMP
- GGS_TRANS_RBA
- GGS_OP_TYPE
- GGS_BEFORE_AFTER_IND

For more information about the @GETENV function, see the *Reference Guide*.

Record data area

The data area of the GoldenGate trail record contains the following:

- The time that the change was written to the GoldenGate file
- The type of database operation
- The length of the record
- The relative byte address within the trail file
- The table name
- The data changes in hex format

The following explains the differences in record image formats used by GoldenGate on Windows, UNIX, Linux, and NonStop systems. The terms “full” and “compressed” image format are used in the descriptions. These terms are used in a different context here than when they are used in other parts of the documentation in reference to how Extract writes column data to the trail, meaning whether only the key and changed columns are written (“compressed”) versus whether all columns are written to the trail (“uncompressed” or “full image”).

Full record image format

Full record image format is only generated in the trail when the source system is HP NonStop, and only when the IOType specified in the record header is one of the following:

3 — Delete
5 — Insert
10 — Update

Each full record image has the same format as if retrieved from a program reading the original file or table directly. For SQL tables, datetime fields, nulls, and other data is written exactly as a program would select it into an application buffer. Although datetime fields are represented internally as an eight-byte timestamp, their external form can be up to 26 bytes expressed as a string. Enscribe records are retrieved as they exist in the original file.

When the operation type is Insert or Update, the image contains the contents of the record *after* the operation (the after image). When the operation type is Delete, the image contains the contents of the record *before* the operation (the before image).

For records generated from an Enscribe database, full record images are output unless the original file has the AUDITCOMPRESS attribute set to ON. When AUDITCOMPRESS is ON, compressed update records are generated whenever the original file receives an update operation. (A full image can be retrieved by the Extract process by using the FETCHCOMPS parameter.)

Compressed record format

By default, trail records written by processes on Windows and UNIX systems are always compressed. The format of a compressed record is as follows:

```
<column index><column length><column data>[...]
```

Where:

- <column index> is the ordinal index of the column within the source table (2 bytes).
- <column length> is the length of the data (2 bytes).
- <column data> is the data, including NULL or VARCHAR length indicators.

Enscribe records written from the NonStop platform may be compressed. The format of a compressed Enscribe record is as follows:

```
<field offset><field length><field value>[...]
```

Where:

- <field offset> is the offset within the original record of the changed value (2 bytes).
- <field length> is the length of the data (2 bytes).
- <field data> is the data, including NULL or VARCHAR length indicators.

The first field in a compressed Enscribe record is the primary or system key.

Tokens area

The trail record also can contain two areas for tokens. One is for internal use and is not documented here, and the other is the user tokens area. User tokens are environment values that are captured and stored in the trail record for replication to target columns or other purposes. If used, these tokens follow the data portion of the record and appear similar to the following when viewed with Logdump:

```
TKN-HOST         : syshq
TKN-GROUP        : EXTORA
TKN-BA_IND       : AFTER
TKN-COMMIT_TS    : 2003-03-24 17:08:59.000000
TKN-POS          : 3604496
TKN-RBA          : 4058
TKN-TABLE        : SOURCE.CUSTOMER
TKN-OPTYPE       : INSERT
TKN-LENGTH       : 57
TKN-TRAN_IND     : BEGIN
```

GoldenGate operation types

The following are some of the GoldenGate operation types. Types may be added as new functionality is added to GoldenGate. For a more updated list, use the SHOW RECTYPE command in the Logdump utility.

Table 17 GoldenGate operation types

Type	Description	Platform
1-Abort	A transaction aborted.	NSK TMF
2-Commit	A transaction committed.	NSK TMF
3-Delete	A record/row was deleted. A Delete record usually contains a full record image. However, if the COMPRESSDELETES parameter was used, then only key columns will be present.	All
4-EndRollback	A database rollback ended	NSK TMF
5-Insert	A record/row was inserted. An Insert record contains a full record image.	All
6-Prepared	A networked transaction has been prepared to commit.	NSK TMF
7-TMF-Shutdown	A TMF shutdown occurred.	NSK TMF
8-TransBegin	No longer used.	NSK TMF
9-TransRelease	No longer used.	NSK TMF

Table 17 GoldenGate operation types (continued)

Type	Description	Platform
10-Update	A record/row was updated. An Update record contains a full record image. Note: If the partition indicator in the record header is 4, then the record is in FieldComp format (see "15-FieldComp") and the update is compressed.	All
11-UpdateComp	A record/row in TMF AuditComp format was updated. In this format, only the changed bytes are present. A 4-byte descriptor in the format of <2-byte offset><2-byte length> precedes each data fragment. The byte offset is the ordinal index of the column within the source table. The length is the length of the data.	NSK TMF
12-FileAlter	An attribute of a database file was altered.	NSK
13-FileCreate	A database file was created.	NSK
14-FilePurge	A database file was deleted.	NSK
15-FieldComp	A row in a SQL table was updated. In this format, only the changed bytes are present. Before images of unchanged columns are not logged by the database. A 4-byte descriptor in the format of <2-byte offset><2-byte length> precedes each data fragment. The byte offset is the ordinal index of the column within the source table. The length is the length of the data. A partition indicator of 4 in the record header indicates FieldComp format.	All
16-FileRename	A file was renamed.	NSK
17-AuxPointer	Contains information about which AUX trails have new data and the location at which to read.	NSK TMF
18-NetworkCommit	A networked transaction committed.	NSK TMF
19-NetworkAbort	A networked transaction was aborted.	NSK TMF
90-(GGS)SQLCol	A column or columns in a SQL table were added, or an attribute changed.	NSK
100-(GGS)Purgedata	All data was removed from the file (PURGEDATA).	NSK
101-(GGS)Purge(File)	A file was purged.	NSK non-TMF
102-(GGS)Create(File)	A file was created. The GoldenGate record contains the file attributes.	NSK non-TMF

Table 17 GoldenGate operation types (continued)

Type	Description	Platform
103-(GGS)Alter(File)	A file was altered. The GoldenGate record contains the altered file attributes.	NSK non-TMF
104-(GGS)Rename(File)	A file was renamed. The GoldenGate record contains the original and new names.	NSK non-TMF
105-(GGS)Setmode	A SETMODE operation was performed. The GoldenGate record contains the SETMODE information.	NSK non-TMF
106-GGSChangeLabel	A CHANGELABEL operation was performed. The GoldenGate record contains the CHANGELABEL information.	NSK non-TMF
107-(GGS)Control	A CONTROL operation was performed. The GoldenGate record contains the CONTROL information.	NSK non-TMF
115 and 117 (GGS)KeyFieldComp(32)	A primary key was updated. The GoldenGate record contains the before image of the key and the after image of the key and the row. The data is in FieldComp format (compressed), meaning that before images of unchanged columns are not logged by the database.	Windows and UNIX
116-LargeObject 116-LOB	Identifies a RAW, BLOB, CLOB, or LOB column. Data of this type is stored across multiple records.	Windows and UNIX
132-(GGS) SequenceOp	Identifies an operation on a sequence.	Windows and UNIX
160 - DDL_Op	Identifies a DDL operation	Windows and UNIX
161- RecordFragment	Identifies part of a large row that must be stored across multiple records (more than just the base record).	Windows and UNIX
200-GGSUnstructured Block 200-BulkIO	A BULKIO operation was performed. The GoldenGate record contains the RAW DP2 block.	NSK non-TMF
201 through 204	These are different types of NonStop trace records. Trace records are used by GoldenGate support analysts. The following are descriptions. ◆ ARTYPE_FILECLOSE_GGS 201 — the source application closed a file that was open for unstructured I/O. Used by Replicat. ◆ ARTYPE_LOGGERTS_GGS 202 — Logger heartbeat record.	NSK non-TMF

Table 17 GoldenGate operation types (continued)

Type	Description	Platform
	◆ ARTYPE_EXTRACTERTS_GGS 203 — unused. ◆ ARTYPE_COLLECTORTS_GGS 204 — unused.	
205-GGSComment	Indicates a comment record created by the Logdump utility. Comment records are created by Logdump at the beginning and end of data that is saved to a file with Logdump's SAVE command.	All
249 through 254	These are different types of NonStop trace records. Trace records are used by GoldenGate support analysts. The following are descriptions. ◆ ARTYPE_LOGGER_ADDED_STATS 249 — a stats record created by Logger when the source application closes its open on Logger (if SENDERSTATS is enabled and stats are written to the logtrail). ◆ ARTYPE_LIBRARY_OPEN 250 — written by BASELIB to show that the application opened a file. ◆ ARTYPE_LIBRARY_CLOSE 251 — written by BASELIB to show that the application closed a file. ◆ ARTYPE_LOGGER_ADDED_OPEN 252 — unused. ◆ ARTYPE_LOGGER_ADDED_CLOSE 253 — unused. ◆ ARTYPE_LOGGER_ADDED_INFO 254 — written by Logger and contains information about the source application that performed the I/O in the subsequent record (if SENDERSTATS is enabled and stats are written to the logtrail). The file name in the trace record is the object file of the application. The trace data has the application process name and the name of the library (if any) that it was running with.	NSK non-TMF

Index