InterBase
Database Operations

Disclaimer

Borland International, Inc. (henceforth, Borland) reserves the right to make changes in specifications and other information contained in this publication without prior notice. The reader should, in all cases, consult Borland to determine whether or not any such changes have been made.

The terms and conditions governing the licensing of InterBase software consist solely of those set forth in the written contracts between Borland and its customers. No representation or other affirmation of fact contained in this publication including, but not limited to, statements regarding capacity, response-time performance, suitability for use, or performance of products described herein shall be deemed to be a warranty by Borland for any purpose, or give rise to any liability by Borland whatsoever.

In no event shall Borland be liable for any incidental, indirect, special, or consequential damages whatsoever (including but not limited to lost profits) arising out of or relating to this publication or the information contained in it, even if Borland has been advised, knew, or should have known of the possibility of such damages.

The software programs described in this document are confidential information and proprietary products of Borland.

Restricted Rights Legend. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subdivision (b) (3) (ii) of the Rights in Technical Data and Computer Software clause at 52.227-7013.

Corporate Headquarters: Borland International Inc., 100 Borland Way, P. O. Box 660001, Scotts Valley, CA 95067-0001, (408) 438-5300. Offices in: Australia, Belgium, Canada, Denmark, France, Germany, Hong Kong, Italy, Japan, Korea, Latin America, Malaysia, Netherlands, New Zealand, Singapore, Spain, Sweden, Taiwan, and United Kingdom.

Software Version: V3.0

Current Printing: October 1993
Documentation Version: v3.0.1

Reprint note

This documentation is a reprint of InterBase V3.0 documentation. It contains most of the information from *InterBase Previous Versions Documentation Corrections* and *InterBase Version* 3.2 *Documentation Corrections* and a new index. For information on features added since InterBase Version V3.0, consult the appropriate release notes.

Table Of Contents

Preface

Who Should Use this Book ix
Using this Book x
Text Conventions xi
Syntax Conventions xii
InterBase Documentation xiii

1 Introduction

Overview 1-1
InterBase Utilities 1-3
For More Information 1-4

2 Database Backup and Recovery

Overview 2-1
Choosing a Backup and Restore Utility 2-3
The gbak Utility 2-3
Your Operating System Utility 2-3
Backing Up a Database 2-4
Backing Up a Database to Tape 2-4
Backing Up a Database in Generic Format 2-5
Backing Up a Database on a Remote System 2-5
Restoring a Database 2-7
Changing Database Characteristics 2-9
Changing the Database Page Size 2-9
Converting a Single-File Database to a Multiple-File Database 2-10
Changing the On Disk Structure 2-11
For More Information 2-12

3 Database Maintenance

Overview 3-1
Sweeping a Database 3-2
Validating a Database 3-3
Repairing a Corrupt Database 3-4
Enabling After-Image Journaling 3-6
Automating Transaction Recovery 3-7
The Two-Phase Commit Process 3-7
Understanding Automated Transaction Recovery 3-7
Using gfix to Recover Transactions 3-8
Automated Transaction Recovery Restrictions 3-9
For More Information 3-10

4 Journaling

Overview 4-1
Components of Journaling 4-2
Using Journaling for Disaster Recovery 4-4
UNIX Journaling 4-5
Starting UNIX Journaling 4-5
Enabling UNIX After-Image Journaling 4-7
Communicating with the Journal Server 4-8
Disabling UNIX Journaling 4-8
Stopping and Restarting UNIX Journaling 4-9
Changing a Journal File 4-10
Using Journaling to Recover From a Disaster 4-10
Recovering from a System Disaster 4-11
Recovering from a Human Disaster 4-12
VMS Journaling 4-14
Customizing the VMS Journaling Environment 4-14
Starting VMS Journaling 4-15
Enabling VMS After-Image Journaling 4-15
Monitoring Journaling Status 4-16
Changing Journal Files 4-17

Disabling VMS Journaling 4-19
Stopping and Restarting Journaling 4-20
Recovering from a System Disaster 4-21
Recreating a Database up to a Journal Checkpoint 4-23
For More Information 4-26

5 Disk Shadowing

Overview 5-1
Components of Disk Shadowing 5-2
Using Disk Shadowing for Disaster Recovery 5-3
Defining a Shadow 5-4
Defining a Multi-File Shadow 5-6
Adding an Additional File to a Shadow 5-7
Deleting a Shadow 5-9
Transaction Considerations 5-10
Activating a Shadow 5-11
Troubleshooting 5-12
For More Information 5-13

6 Database Operations Reference

Overview 6-1
Gbak 6-2
Gcon 6-10
Gcsu 6-14
Gfix 6-18
Gltj 6-24
Grst 6-26

Preface

This book contains information on the InterBase database utilities.

Who Should Use this Book

The audience for the *Database Operations* manual is a database administrator or user who wants to know how to maintain InterBase databases. This book assumes some database knowledge, but no knowledge of how to use any of the InterBase utilities.

Using this Book

This book contains the following chapters:

Chapter 1	Introduces you to all of the InterBase database utilities.
Chapter 2	Describes how to use **gbak** to backup and restore databases.
Chapter 3	Explains how to use **gfix** to perform database maintenance.
Chapter 4	Explains how to use journaling on UNIX and VMS systems.
Chapter 5	Describes how to set up a disk shadow for a database.
Chapter 6	Contains reference information on the InterBase utilities.

Text Conventions

This book uses the following text conventions.

boldface

Indicates a command, option, statement, or utility. For example:

- Use the **commit** command to save your changes.
- Use the **sort** option to specify record return order.
- The **case_menu** statement displays a menu in the forms window.
- Use **gdef** to extract a data definition.

italic

Indicates chapter and manual titles; identifies filenames and pathnames. Also used for emphasis, or to introduce new terms. For example:

- See the introduction to SQL in the *Programmer's Guide*.
- /usr/interbase/lock_header
- Subscripts in RSE references *must* be closed by parentheses and separated by commas.
- C permits only *zero-based* array subscript references.

`fixed width font`

Indicates user-supplied values and example code:

- `$run sys$system:iscinstall`
- `add field population_1950 long`

UPPER CASE

Indicates relation names and field names:

- Secure the RDB$SECURITY_CLASSES system relation.
- Define a missing value of *X* for the LATITUDE_COMPASS field.

Syntax Conventions

This book uses the following syntax conventions.

{braces}	Indicates an alternative item: • option::= {**vertical** | **horizontal** | **transparent**}
[brackets]	Indicates an optional item: • dbfield-expression[**not**]**missing**
`fixed width font`	Indicates user-supplied values and example code: • $run sys$system:iscinstall • add field population_1950 long
commalist	Indicates that the preceding word can be repeated to create an expression of one or more words, with each word pair separated by one comma and one or more spaces. For example, field_def-commalist resolves to: field_def[,field_def[,field_def]...]
italics	Indicates a syntax variable: **create_blob** *blob-variable* **in** *dbfield-expression*
|	Separates items in a list of choices.
⇓	Indicates that parts of a program or statement have been omitted.

InterBase Documentation

The InterBase Version 3.0 documentation set contains the following books:

Getting Started with InterBase (INT0032WW2179A) provides an overview of InterBase components and interfaces.

Database Operations (INT0032WW2178D) describes how to use InterBase utilities to maintain databases.

Data Definition Guide (INT0032WW2178F) describes how to create and modify InterBase databases.

DDL Reference (INT0032WW2178E) describes the function and syntax for each of the data definition language clauses and statements. It also lists the standard error messages for **gdef**.

DSQL Programmer's Guide (INT0032WW2179C) describes how to program with DSQL, a capability for accepting or generating SQL statements at runtime.

Forms Guide (INT0032WW2178A) describes how to create forms using the InterBase forms editor, **fred,** and how to use forms in **qli** and GDML applications.

Programmer's Guide (INT0032WW2178I) describes how to program with GDML, a relational data manipulation language, and SQL, an industry standard language.

Programmer's Reference (INT0032WW2178H) describes the function and syntax for each of the GDML and InterBase supported SQL clauses and statements. It also lists the standard error messages for **gpre**.

Qli Guide (INT0032WW2178C) describes the use of **qli**, the InterBase query language interpreter that allows you to read to and write from the database using interactive GDML or SQL statements.

Qli Reference (INT0032WW2178B) describes the function and syntax for each of the data definition, GDML, and SQL clauses and statements that you can use in **qli.**

Sample Programs (INT0032WW2178G) contains sample programs that show the use of InterBase features.

Master Index (INT0032WW2179B) contains index entries for the entire InterBase Version 3.0 documentation set.

Chapter 1
Introduction

This chapter provides a brief introduction to the InterBase utilities.

Overview

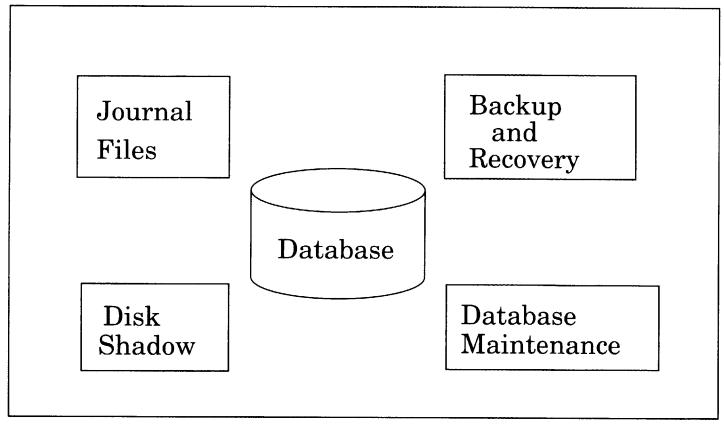
This book describes how to perform tasks involved with the administration and operation of an InterBase database. InterBase does not require a database administrator, and all of the tasks described in this book can be carried out by any individual who is familiar with InterBase.

Database administration and operations involves doing the following tasks:

- Backup and recovery
- Routine database maintenance
- Preparation for disaster recovery using after-image journaling or disk shadowing

Figure 1-1 shows how database administration and operations can be performed independently from your database application.

Figure 1-1. Database Administration and Operations



InterBase Utilities

InterBase provides the following utilities to perform database administration and operations:

- **gbak**, the backup and restore utility
- **gcon**, the console program used in communicating with the journal server
- **gcsu**, the central servers management utility
- **gfix**, the general database maintenance utility
- **gltj**, the journal server utility
- **grst**, the journal file restoration utility

For More Information

For more information on:

- **gbak**, see Chapter 2, *Database Backup and Recovery*
- **gfix**, see Chapter 3, *Database Maintenance*
- Journaling, **grst**, **gcon**, and **gltj**, see Chapter 4, *Journaling*
- Disk shadowing, see Chapter 5, *Disk Shadowing*

For more information on each utility's syntax and the options, see the entry for that utility in Chapter 6, *Database Operations Reference*.

Chapter 2
Database Backup and Recovery

This chapter describes **gbak**, the utility used for backing up and restoring InterBase databases.

Overview

You should back up your databases on a regular basis, either to a removable medium or to another device. That way, in the event your drive ever crashes, your work up to the last backup is protected. You should also consider storing your backup medium off premises.

InterBase provides a backup and restore utility, **gbak**, for performing these tasks. **gbak** allows you to:

- Backup a database
- Backup a database in a generic (XDR) format
- Restore a database
- Transfer databases between different machine types

- Reconstruct databases restored from the after-image journal
- Change the size of database pages from the default of 1024 bytes up to 8192 bytes
- Change the database from a single-file to a multiple-file database
- Eliminate obsolete versions of a database record
- Balance indexes
- Correct selectivity on non-unique indexes
- Convert databases from one on-disk structure (ODS) to another

The following sections discuss these uses.

Choosing a Backup and Restore Utility

InterBase allows two different methods for backing up and restoring database files:

- **gbak**, a database-specific backup and restore utility.
- Your operating system's backup and restore utility. These utilities include: BACKUP on VMS systems, **tar** on UNIX systems, and WBAK on Apollo systems.

The advantages of each are discussed below.

The gbak Utility

gbak offers the following advantages:

- *Backups can run concurrently with other users.* **gbak** does not require exclusive access the database files it is copying. You can use **gbak** to create a database backup, and then include the database backup as part of your regular system backup.
- *Multi-file databases are never partially backed up.* If your database spans multi-files, **gbak** does not do a partial backup of your database. It either backs up all of your database files, or none at all.
- *Ability to backup and restore data from one operating system to another operating system.* **gbak** backs up and restores data from one operating system to another with or without a network connection. It automatically translates the data from one operating system to another operating system.
- *Improves database performance.* **gbak** can improve your database performance, because it improves the locality of reference, record density, and index balance. In addition, you can use **gbak** to change the database page size from the default of 1024 bytes up to 8192 bytes, or to split a single-file database into a multi-file database.

Your Operating System Utility

Your operating system backup and restore utility offers only one advantage over **gbak**—backups are usually faster.

Backing Up a Database

gbak lets you backup and restore data from one operating system to another. You can backup the database file onto a cartridge tape, a magnetic tape, or directly onto a device. The device can be either local or remote.

Note

Backing up onto cartridge tape and magnetic tape is supported on Apollo and UNIX systems.

Backing up and restoring a database from one operating system to another requires a usually requires network connection between the two machines, unless you use the **gbak**'s **transportable** option. The InterBase access method takes care of incompatibilities in storage formats between the two operating systems.

If you do not have a network connection between the two machines you want to back up and restore a database on, you can build a generic backup using **gbak** 's **transportable** option. Transportable **gbak** is discussed later in this chapter.

The following sections provide examples of backing up a database on a local or a remote device onto different types of media.

Backing Up a Database to Tape

The following command uses **gbak** to back up a database to a file on another device:

O/S	Command
Apollo AEGIS	% gbak atlas.gdb atlas_01_march.gbak -b
UNIX	% gbak -b atlas.gdb atlas_01_march.gbak
VMS	$ gbak/backup atlas.gdb atlas_01_march.gbak

The following example shows how to backup the sample database on an Apollo system onto a local cartridge tape or magnetic tape:

O/S	Command
Apollo AEGIS	% gbak atlas.gdb -b -dev ct

If you do not specify a device, **gbak** defaults to a magnetic tape.

The following example backs up only the data definition. You specify this option if you want an empty version of an existing database:

O/S	Command
Apollo AEGIS	% gbak atlas.gdb atlas_01_march.gbak -b -m
UNIX	% gbak -b -m atlas.gdb atlas_01_march.gbak
VMS	$ gbak/backup/metadata atlas.gdb atlas_01_mar.gbak

Backing Up a Database in Generic Format

If you do not have a network connection between the two machines, and you want to backup a database on one type of machine and restore it on another type of machine, you can use the **transportable** option. This option writes data in a generic (XDR) format, which can be restored on any type of machine.

When you use the **transportable** option, you should consider the way in which the machines you are backing up and restoring the database on handle security. If you are not restricted by security, you must change the database security after you restore it, so it corresponds to the security scheme on the new machine.

The following example writes the data from the atlas database on an Apollo in generic format without the database's security:

```
% gbak -t /interbase/examples/atlas.gdb  atlas_t.gbak
```

The *atlas_t.gbak* file can be restored onto another machine with an incompatible storage format:

```
% gbak -c atlas_t.gbak  new_atlas.gdb
```

Backing Up a Database on a Remote System

If you are using InterBase in a heterogeneous environment, you can use **gbak** to move a database or just the data definition from SUN to VMS, ULTRIX, HP, or Apollo. InterBase provides full read-write access to databases in a heterogeneous network of VAXes, SUNs, Apollos, and HPs. You might move a database onto another machine, rather than access it remotely in order to:

- Have access to larger and faster disks
- Put the database on the machine where it is used most frequently

Table 2-1 lists how to access a remote database.

Table 2-1. Remote Database Access

From	To	Syntax
VMS	VMS via DECnet	node-name::filespec
VMS	ULTRIX via DECnet	node-name::filespec
VMS	non-VMS and non-ULTRIX	node-name^filespec
ULTRIX	VMS via DECnet	node-name::filespec
Apollo	Apollo	//node-name/filespec
Everything Else	Whatever is left	node-name:filespec

Note

If you are running on a UNIX system, you must enclose the remote database name in single quotes.

The following example, run on a MicroVAX with ULTRIX-32m, backs up a database on an Apollo workstation to a file on the MicroVAX, and restores the database for use on the MicroVAX:

```
$ gbak apollo:/interbase/examples/atlas.gdb  atlas.gbak
$ gbak -r atlas.gbak  atlas.gdb
```

The database is now ready to use on the MicroVAX:

```
$ qli
Welcome to QLI
Query Language Interpreter
QLI> ready atlas.gdb
       ⇓
QLI> exit
$
```

If you want to copy the data definition for the database without the data, you can use **gbak**'s **metadata** switch. The following example backs up and restores the metadata for the atlas database:

```
$ gbak -m apollo:/interbase/examples/atlas.gdb  atlas.gbak
$ gbak -r atlas.gbak  atlas.gdb
```

Restoring a Database

gbak restores databases from backup copies. The following command restores the database from the backup copy, creating a new database file:

O/S	Command
Apollo AEGIS	% gbak atlas_01_march.gbak atlas.gdb -c
UNIX	% gbak -c atlas_01_march.gbak atlas.gdb
VMS	$ gbak/create atlas_01_march.gbak atlas.gdb

The **create** option restores a database from the backup file to a new file. If the file *new_atlas.gdb* already exists on:

- VMS, a new version of the file is created
- UNIX and Apollo systems, the restoration fails because **gbak** only overwrites existing files if you use the **replace** switch

To view a description of what happens during a backup or restoration, you can include the **verify** option. This switch provides detailed information about the operation:

O/S	Command
Apollo AEGIS	% gbak atlas.gdb atlas.gbak -c -v
UNIX	% gbak -c -v atlas.gdb atlas.gbak
VMS	$ gbak/create/verify atlas.gdb atlas.gbak

```
readied database atlas.gdb for backup
creating file atlas.gbak
starting transaction
writing global fields
     writing global field STATE
     writing global field POPULATION
      ⇓
writing relations
   writing relation STATES
    writing field STATEHOOD
    writing field AREA
         ⇓
```

```
writing data for relation CANADIAN_TOURISM
12 records written
writing data for relation PROVINCES
12 records written
        ⇓
closing file, committing, and finishing
```

The following example restores a database without its validity conditions. You use the **no_validity** option if you want to restore a database without **valid_if** clauses:

O/S	Command
Apollo AEGIS	% gbak atlas.gdb new_atlas.gbak -b -n
UNIX	% gbak -b -n atlas.gdb new_atlas.gbak
VMS	$ gbak/backup/no_validity atlas.gdb new_atlas.gbak

Note

When you back up a database that includes fields which are computed or validated through user defined functions (UDFs), you must have the UDF function library available when you restore the backup. During a restore, function libraries are needed to validate metadata.

If a required function library is not available, you will get an error message when you attempt to restore the data. If the source of the problem is not clear, rerun the **gbak** restore with the -o and -v options, which provide more information. A database recreated with the -o option includes an RDB$FUNCTIONS relation. RDB$FUNCTIONS contains the library name for each function.

Changing Database Characteristics

You can use **gbak** to change the following database characteristics:

- Database page size
- Single-file database into a multiple-file database

Each of these is discussed in this section.

Changing the Database Page Size

You may want to try to improve performance by changing the page size of a database. Each InterBase index bucket is one database page long. Making longer database pages mean larger buckets and fewer levels in the hierarchy.

You can use **gbak** to change the default database page size from 1024 bytes. Interbase supports database page sizes of 1024, 2048, 4096, and 8192. When determining what size database page you should choose for your application, consider the following guidelines:

- Blob storage and retrieval is most efficient when the entire blob fits on a single page. If the application stores blobs between 1K and 2K, using a database page size of 2K reduces blob access time. If your application has an occasional blob of 2K, you should not increase the database page size.
- InterBase performs better if each record fits on a page. Calculating the amount of space used on a page depends on the amount of compression that can be performed on the record. A database containing long records with non-repetitive data performs better with a larger page size.
- Index performance improves if the depth of the index is kept to a minimum. Doubling the database page size doubles the number of nodes the first level of the index points to, thus doubling the scope of the second level.

The following examples change the default database page size from 1024 pages to 2048 pages:

1. Backup your database without specifying the **page_size** option. For example:

O/S	Command
Apollo AEGIS	% gbak atlas.gdb atlas_01_march.gbak -b
UNIX	% gbak -b atlas.gdb atlas_01_march.gbak
VMS	$ gbak/backup atlas.gdb atlas_01_march.gbak

2. Restore your database using the **page_size** option. For example:

O/S	Command
Apollo AEGIS	% gbak atlas.gbak new_atlas.gdb -r -p 2048
UNIX	% gbak -r -p 2048 atlas.gbak new_atlas.gdb
VMS	$ gbak/replace/page_size 2048 atlas.gbak new_atlas.gdb

Your database now has a 2048 page size.

Converting a Single-File Database to a Multiple-File Database

You can change a database from a single-file database to a multiple-file database with **gbak**. By default, InterBase stores and backs up databases into a single-file database. If your database grows to a very large size, you may want to split the database up into multiple database files. When using **gbak** to restore your database into multiple-files, be sure you specify a pathname for each file.

The following example restores a single-file database into a multiple-file database:

O/S	Command
Apollo AEGIS	% gbak atlas.gbak atlas.gdb 100 atlas1.gdb -r
UNIX	% gbak -r atlas.gbak atlas.gdb length 100 atlas1.gdb
VMS	$ gbak/replace atlas.gbak atlas.gd 100 atlas1.gdb

You can also use the **define database** and **modify database** statements to create a multiple-file database. For more information on these statements, see the *DDL Reference*.

Changing the On Disk Structure

You can use **gbak** to convert from one on disk structure (ODS) to another. This is only necessary if InterBase changes the on disk structure. When InterBase changes the on disk structure, we ship a special image called the *bridge*. This image can read Version 2.*n* and Version 3.0 databases.

Note

If you are upgrading from Version 2.*n* to Version 3.0, you must use **gbak** to convert to the new on disk structure to take advantage of the new Version 3.0 features.

To convert to the new on disk structure:

1. Backup your current database using **gbak**. For example:

O/S	Command
Apollo AEGIS	% gbak atlas.gdb v2_atlas.gbak -b
UNIX	% gbak -b atlas.gdb v2_atlas.gbak
VMS	$ gbak/backup atlas.gdb v2_atlas.gbak

2. Restore your database using **gbak**. For example:

O/S	Command
Apollo AEGIS	% gbak v2_atlas.gbak v3_atlas.gdb -c
UNIX	% gbak -c v2_atlas.gbak v3_atlas.gdb
VMS	$ gbak/create v2_atlas.gbak v3_atlas.gdb

For more information on converting to the new ODS, see your platform-specific installation instructions.

For More Information

For more information on **gbak**'s syntax and options, see the entry on **gbak** in Chapter 6, *Database Operations Reference*.

For more information on creating and modifying a database, see the chapter on creating a database in the *Data Definition Guide*.

For more information on the **define database** and **modify database** statements, see the entries for these statements in the *DDL Reference*.

Chapter 3
Database Maintenance

This chapter describes **gfix**, the InterBase utility used for minor system maintenance.

Overview

Occasionally a database incurs minor damage from a disk controller or an operating system write error. Such an error might result in a "broken" or "lost" data structure, such as a database page or index.

gfix is the system utility that fixes minor database problems such as these. In addition, **gfix**:

- Sweeps a database to release storage space by cleaning up old record versions
- Validates databases
- Enables after-image journaling
- Prevents limbo transactions by providing automated transaction recovery

Each of these topics is discussed in the following sections.

Sweeping a Database

You can use **gfix** to sweep a database without taking the database offline. When **gfix** sweeps a database it cleans up old record versions. This is called, *garbage collection*. Although the InterBase access method automatically performs some garbage collection when it visits a record, it does not necessarily release all the space associated with the removed record version. **gfix** visits all records in a relation and releases space held by:

- Records that were rolled back
- Out of date record versions

You can release space held by records that were rolled back or by out of date record versions by using either the **housekeeping** option or the **sweep** option.

The **gfix housekeeping** option lets you set the automatic sweep interval to any non-negative integer. Currently, the access method automatically sweeps a database every 20,000 transactions. You may want to change the sweep interval to be smaller or larger, depending on how often you want to clean up old record version. To disable this option, use **gfix** to set the sweep interval to 0.

If you want to sweep a database immediately, you can use the **gfix sweep** option.

The following command sweeps a database for old records every 10,000 transactions. Because you can sweep a database without taking it offline, you can run **gfix** in the background at a low priority:

O/S	Command
Apollo AEGIS	% gfix atlas.gdb -h 10000
UNIX	% gfix -h 10000 atlas.gdb
VMS	$ gfix/housekeeping 10000 atlas.gdb

Validating a Database

You can use **gfix** to validate your database structures to verify their integrity. To do this, you use the **gfix -validate** option. This option:

- Requires *exclusive* access to the database
- Frees up pages that have been assigned to any database structure, but not used
- Provides options for repairing a corrupt database:
  - **full**, verifies record and page structures, releasing record fragments not currently assigned to current structures
  - **no_update**, validates and reports corrupt and misallocated database structures, but *does not* fix them
  - **mend**, marks records as corrupt structures

Some of the errors **gfix** finds with the **validate** option are normal ones that can occur if a program aborts. When a program aborts, no committed data is lost and uncommitted updates are rolled back. However, the InterBase access method may have assigned a page from free space for the records. The **validate** option reports such pages as *orphan pages* and assigns them to free space.

The following example uses the **validate** and **full** options:

O/S	Command
Apollo AEGIS	% gfix atlas.gdb -v -f
UNIX	% gfix -v -f atlas.gdb
VMS	$ gfix/validate/full atlas.gdb

The **validate** option also locates problems caused by write errors in the operating system or hardware. These errors can make committed data unrecoverable and cannot be corrected with the **validate** option. Use **gfix**'s **mend** option to correct such errors. This option marks records as corrupt structures, thus causing the access method to skip them.

Repairing a Corrupt Database

Follow the procedure described below if you suspect that you have a corrupt database:

1. Make a copy of the database with an operating system utility. Do not use **gbak**. For example:

O/S	Command
Apollo AEGIS	% cpf atlas.gdb broken_atlas.gdb
UNIX	% cp atlas.gdb broken_atlas.gdb
VMS	$ copy atlas.gdb broken_atlas.gdb

2. Run **gfix -mend** on the copy of your database. **gfix** reports the errors it finds. For example:

O/S	Command
Apollo AEGIS	% gfix broken_atlas.gdb -m
UNIX	% gfix -m broken_atlas.gdb
VMS	$ gfix/mend broken_atlas.gdb

3. Run **gfix -validate** to see if errors **gfix** reported were fixed. For example:

O/S	Command
Apollo AEGIS	% gfix broken_atlas.gdb -v
UNIX	% gfix -v broken_atlas.gdb
VMS	$ gfix/validate broken_atlas.gdb

4. Run **gbak** to backup the mended database. For example:

O/S	Command
Apollo AEGIS	% gbak broken_atlas.gdb broken_atlas.gbak -b
UNIX	% gbak -b broken_atlas.gdb broken_atlas.gbak
VMS	$ gbak/backup broken_atlas.gdb broken_atlas.gbak

5. Run **gbak** to restore the database. This rebuilds the indexes and other database structures. For example:

O/S	Command
Apollo AEGIS	% gbak broken_atlas.gbak new_atlas.gdb -r
UNIX	% gbak -r broken_atlas.gbak new_atlas.gdb
VMS	$ gbak/create broken_atlas.gbak new_atlas.gdb

There are some types of database corruption **gfix** cannot fix. If you encounter a corrupt database and **gfix** is unable to repair it, send the copy of the pre-**gfix** database you made in Step 1 to:

Borland International Inc.
InterBase Support
1800 Green Hills Road
P. O. 660001
Scotts Valley, CA 95067

You can also call Customer Support at 800-437-7367 from anywhere in the United States and Canada or at 408-431-5400 from outside of the United States. In addition, you can fax a description of the problem to 408-439-7808.

Enabling After-Image Journaling

You can use **gfix** to enable after-image journaling for a database. The following example enables journaling for the atlas database.

Note

This example assumes you have set up a journal directory, created the journal database, and started the journal server.

O/S	Command
Apollo AEGIS	% gfix atlas.gdb -e
UNIX	% gfix -e atlas.gdb
VMS	$ gfix/enable atlas.gdb

Automating Transaction Recovery

To understand how to use **gfix** for automated transaction recovery, you must first understand the InterBase two-phase commit process. This process is described below, followed by an overview of automated transaction recovery.

The Two-Phase Commit Process

The *two-phase commit* process occurs for transactions that span multiple databases. This process virtually guarantees that unless there is a catastrophic event, updates to multiple databases within a single transaction happen either in parallel or not at all. When a user commits a multiple database transaction, InterBase performs a two-phase commit automatically.

The two-phase commit works the following way:

- In phase 1, InterBase prepares each database for the commit. This involves completing each *subtransaction* by writing the changes to disk. A subtransaction is the component of a multi-database transaction that involves a single database.
- In phase 2, InterBase marks each subtransaction as committed. It marks them in the same order in which they were prepared.

If there is a network failure, disk crash, or other catastrophic event that makes one or more databases unavailable during the two-phase commit process, the commit will fail. If this happens during phase 2 of the commit process, some of the subtransactions will be committed and others will not.

Understanding Automated Transaction Recovery

gfix provides an automated method for recovering from the failure of a two-phase commit. This utility reconnects all subtransactions, analyzes their state, provides advice, and does a commit or rollback, as requested.

gfix analyzes the state of the subtransactions by figuring out when the two-phase commit failed:

- If the first transactions are in limbo and later transactions are not, **gfix** assumes that the prepare phase did not complete. In this case, **gfix** advises you to do a rollback.
- If the first transactions are missing and later transactions are in limbo, **gfix** assumes that the prepare phase completed and the commit phase did not. In this case, **gfix** advises you to commit the transaction.

- If all transactions are prepared, **gfix** assumes that the failure occurred between phase 1 and phase 2 of the two-phase commit process. In this case, the **gfix** users must decide whether to commit or roll back the transaction.

Using gfix to Recover Transactions

Use **gfix** to recover from the failure of a two-phase commit. Several of **gfix**'s options have been extended in Version 3.0 to include multiple database functionality. Table 3-1 describes these options and the new **two_phase** option.

Table 3-1. gfix Options for Two-Phase Commit

Option	Meaning
list	Prints all transactions in limbo, the partner, the current state, and action automated transaction would take for multi-database transactions.
prompt	Prompts the user for advice on how to complete the two-phase commit process.
commit	Commits all transactions and the partners of the transaction. **gfix** advises against a commit if some of the transactions were not prepared.
rollback	Rolls back all transactions and the partners of the transaction
two-phase	Depending on the state of the transaction, **gfix** commits or rolls back the transaction to preserve the two-phase commit. This option prompts you for advice if all transactions are prepared and you did not specify a commit or a rollback

To recover from a catastrophic event using automatic transaction recovery, invoke **gfix** with the **-two_phase** and **-list** options. **gfix** prints the current state of the transactions and their partners and does an automatic commit or rollback.

Automated Transaction Recovery Restrictions

Automated transaction recovery can only be used against transactions prepared by InterBase in the course of a commit. A user-defined two-phase commit using a separate **prepare** statement cannot use automated transaction recovery. If you want to use your own two-phase recovery, you can still use **gfix** to recover transactions in limbo. However, these transactions are treated as a single database transaction, and each transaction has to be manually recovered.

For More Information

For more information on **gfix**'s syntax and options, see the entry on **gfix** in Chapter 6, *Database Operations Reference*.

If you find that **gfix** is not suited to your application, you can write a routine or your own utility. **gfix** calls the following **gds** routines for most of its functions:

- **gds_$attach_database** to attach the database. Options on the call provide the **verify**, **sweep**, and **mend** functions.
- **gds_$transaction_info** to execute the listing option.
- **gds_$reconnect_transaction** to execute the commit and rollback options.

See the *OSRI Guide* for more information about using these routines.

Chapter 4
Journaling

This chapter describes after-image journaling on UNIX and VMS systems.

Overview

After-image journaling or *long-term journaling* allows for the recovery of data in the event the original database file becomes unreadable. After-image journaling is an optional part of InterBase. Once you enable journaling, changes to the database are automatically written to a journal file. The journal file can be located on tape or disk, depending on the type of system.

InterBase supports journaling on Apollo, UNIX, and VMS systems. Apollo and UNIX support journaling to disk and VMS supports journaling to tape. When you journal to a disk, be aware of the following guidelines on each of the hardware platforms:

- Apollo supports journaling to any disk on the ring.
- Sun supports journaling on machines running the same version of SunOS and of the same architecture type. You cannot journal across different architectures or different versions of SunOS.
- UNIX supports journaling to any NFS mounted disk.
- VMS supports journaling on any disk in a cluster. You cannot journal over DECnet.

Components of Journaling

After-image journaling has the following components:

- **gltj**, the journal server that starts and controls journaling. You can run this server as a detached background process.
- **Gcon**, the utility that connects to the journal server (**gltj**). It allows you to modify journaling status. **Gcon** is supported only on UNIX.
- **Gfix**, the means by which you enable and disable journaling for a particular database and its journal directory.
- A journal database, journal.gdb, maintained by **gltj**. It contains the names of databases for which journaling is enabled and the names of the files to which journal records are written.
- Permanent journal files that record database changes. These files can be located on disk or tape. For each journal directory there is only one active permanent journal file. Other journal files can be queued and started automatically when the current one is closed.
- **grst**, the database restore utility that restores journal files.
- A journal directory, containing the journal.gdb database and the intermediate journal file *interbas.jrn*. The *interbas.jrn* journal file is only used on VMS systems. This directory should be on a different device from the database to avoid a simultaneous loss of the intermediate journal and the database.
- *Interbas.jrn*, the intermediate journal file. When programs update journaled databases, their changes are automatically written to this file. **gltj** monitors *interbas.jrn* and writes blocks of changes made to that file to the permanent journal files. The *interbase.jrn* file is used only in VMS journaling.

The following figures diagram after-image journaling components. Figure 4-1 diagrams journaling on UNIX systems, and Figure 4-2 diagrams journaling on VMS systems.

Figure 4-1. UNIX Journaling Diagram

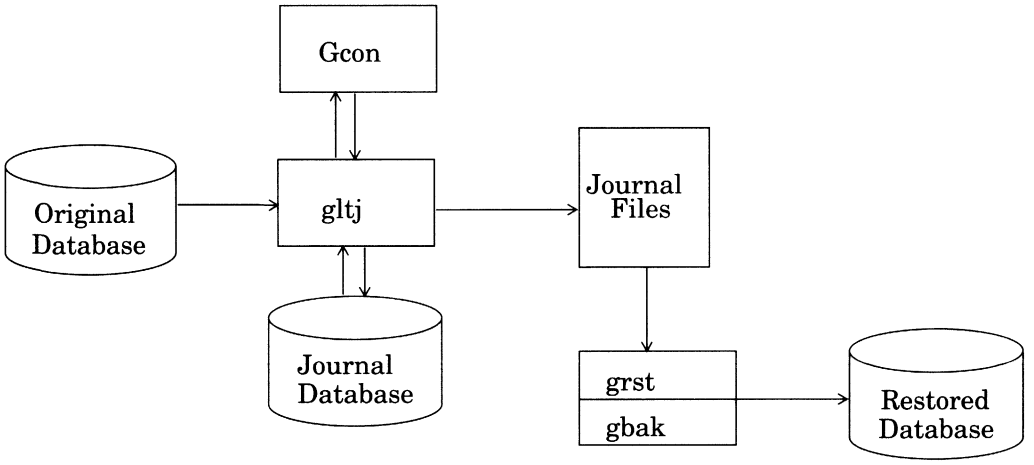
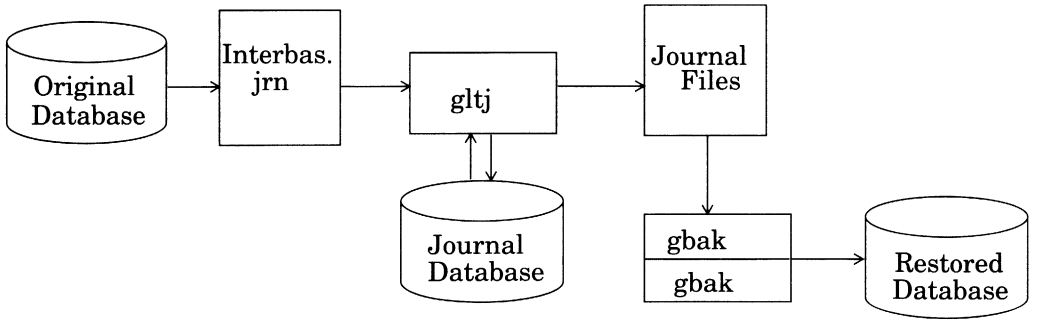


Figure 4-2. VMS Journaling Diagram



There are several steps involved in starting, enabling, and disabling journaling UNIX and VMS systems. These steps are described later in this chapter.

Using Journaling for Disaster Recovery

InterBase provides after-image journaling as one means of disaster recovery. After-image journaling has some advantages and disadvantages.

The advantages of after-image journaling are:

- Recovery from user media errors. You can recover a database saved at a previous time to a journal file.
- Less online storage space.

Journaling has a few disadvantages. When considering journaling as the method for disaster recovery you should consider the following disadvantages:

- Journal files can grow indefinitely. This requires an operator to archive these files on removable storage.
- Starting journaling requires exclusive access to the database.
- Recovering from a disaster using journal files requires rebuilding the database. This can be time consuming.
- Journaling requires an extra process to start and manage the journal server.
- Setting up and monitoring journaling is a moderately complex process.
- If the current journal file fills up, users hang until another journal file is made available.

UNIX Journaling

The next several sections describe how to start, enable, and disable journaling, and how to restore a database from an after-image journal file on Apollo AEGIS, Apollo DOMAIN, and UNIX systems.

Note

If you use an Apollo AEGIS shell, use the AEGIS syntax for entering the after-image journaling commands, otherwise use the UNIX syntax.

Starting UNIX Journaling

Before you can enable journaling for a database, you must:

- Set up the journal directory
- Create the journal database
- Start the journal server, **gltj**

  You can start **gltj** as a separate background process. Once you start **gltj**, it remains available until the process dies or the system crashes. You can also start the journal server in a system startup file.

Note

You can run the journal server in a window. To do this, use the **gltj -debug** command. The **-debug** option signals to the **gltj** process it is running in a window.

When you start the journal server, it automatically creates two files, *console.addr* and *journal.addr*, for establishing TCP/IP connections. It also automatically creates a *journal.log* file. This file keeps a log of the messages sent to **gltj** from standard output and standard error.

To set up your journal directory, start **gltj**:

1. Create a journal directory and change to that directory. The following command creates a journal directory, and makes it the default directory

O/S	Command
Apollo AEGIS	% crd /usr/journal % wd /usr/journal

O/S	Command
UNIX	% mkdir /usr/journal % cd /usr/journal

2. Create a journal database in the journal directory. The installation procedure leaves a backup copy of the journal database in */interbase/com/journal.gbak* on Apollo systems and in */usr/interbase/bin/journal.gbak* on UNIX systems. The following command uses **gbak** to create a journal.gdb database:

O/S	Command
Apollo AEGIS	% gbak /interbase/com/journal.gbak journal.gdb -c
UNIX	% gbak -c /usr/interbase/bin/journal.gbak journal.gdb

3. Start **gltj** as a background process and pipe its output to a file. The following command starts the journal server in a C shell as a background process:

```
% gltj &
[1] 346
```

If you want to verify that the journal server process is running, use the UNIX command **ps -ax** if you are using Berkeley UNIX. Use the UNIX **ps -e** command if you are using System V UNIX. For example:

```
% ps -ax
3976 ttyp2    0:00 gltj
```

4. Invoke **gcon** and queue one or more journal files. Note that **gcon** uses gltj> as its prompt. The following example queues two journal files:

```
%gcon
08:25:47 03/01/90 Console connected
Journaling disabled; no output file
gltj> queue journ.a
gltj> queue journ.b
```

5. Check the status of journaling:

```
gltj> status
Journaling enabled; current output file: journ.a
Output files queued:
        journ.b
gltj>
```

6. Exit from **gcon**:

```
gltj> exit
%
```

The following section explains how to enable after-image journaling.

Enabling UNIX After-Image Journaling

When you enable journaling for a database, InterBase automatically backs up the database into the journal files. This way you do not have to perform a separate backup or locate the right backup when you need to recover the database.

To enable journaling on UNIX:

1. Use **gfix** to enable journaling for a database. Enable the journaling in the same directory you started **gltj** from. *Enabling journaling requires exclusive access to the database.* If users are active on the database, the **gfix** command *does not* complete until the other users exit from the database. The following command enables journaling for the atlas.gdb database:

O/S	Command
Apollo AEGIS	% gfix /interbase/journal_dir atlas.gdb -enable
UNIX	% gfix -enable /usr/interbase/journal_dir atlas.gdb

2. Invoke **gcon** and check the status of journaling:

```
% gcon
08:25:47 03/01/90 Console connected
Journaling enabled; current output file: journ.a
Output files queued:
        journ.b
Known databases and connections:
      /usr/interbase/journal.dir/atlas.gdb (1), connections: 0
gltj>
```

Once you enable journaling, InterBase tries to establish a connection to **gltj**, the journal server. You must keep **gltj** running as long as you want to journal changes to the database. If the **gltj** process dies or the system crashes, you have to restart it as a separate background process. Otherwise, the process returns an error whenever someone tries to attach a database it is journaling.

Communicating with the Journal Server

At any point after you have enabled after-image journaling, you may want to check on its status. To do this, use the **gcon** console program utility. It allows you to communicate directly with the journal server process.

Although you can have multiple **gcon** processes running and communicating with the journal server, it is not recommended. Like the journal server, **gcon** must be started from the same directory as the journal files.

Note

If you are running journaling in a network, you can rlogin to the journal directory on UNIX systems, or you can use the **set host** command on VMS systems.

Table 4-1 lists **gcon**'s options and their meanings.

*Table 4-1. **Gcon**'s Options and Their Meanings*

Option	Meaning
queue	Queues a file or device for journaling
close	Closes the current journal file or device
status	Displays the information about the journal server process
shutdown	Shuts down the journal server process
suspend	Interrupts journaling
resume	Continues journaling after it has been suspended
help	Displays online help for gcon
version	Displays the software version number
exit	Exits from the gcon utility

Disabling UNIX Journaling

When you disable journaling for a database, the journal server must be running. If you try to disable a journal database and the journal server is not running, **gfix** returns an error message and does not disable journaling.

To disable journaling on UNIX:

1. Use **gfix** to disable journaling for a database. The following command disables journaling for the atlas.gdb database:

O/S	Command
Apollo AEGIS	% gfix /interbase/atlas.gdb -disable
UNIX	% gfix -disable /usr/interbase/atlas.gdb

2. Check the status of journaling:

```
% gcon
08:25:47 03/01/90 Console connected
Journaling disabled; current output file: journ.a
Output files queued:
        journ.b
Known databases and connections:
      /usr/interbase/journal.dir/atlas.gdb (1), connections: 0
gltj>
```

After you disable journaling for a database, the journal process remains known to the journal server.

Stopping and Restarting UNIX Journaling

To stop after-image journaling, use the **gcon shutdown** command. This command shuts down the journal server process.

To restart journaling, re-invoke the journal server from the same directory you stopped journaling in. Use the **gcon resume** command to resume after-image journaling.

To stop and restart UNIX journaling:

1. Invoke **gcon** and shut down the journal server process. For example:

```
% gcon
gltj> shutdown
%
```

2. Restart the journal server process. For example:

```
% gltj  &
[1] 9296
```

3. Invoke **gcon** and resume journaling. For example:

```
% gcon
Journal server console program
Journaling disabled; no output file
Known databases and connections:
       /usr/interbase/journal.dir/atlas.gdb (2), connections: 0
gltj> resume
```

4. Exit from **gcon**.

Changing a Journal File

Occasionally, you may need to close one permanent journal file and open another one.

To change to another journal file:

1. Invoke **gcon** and queue a new permanent journal file. For example:

```
% gcon
gltj> queue journ.c
gltj>
```

2. Close the current permanent journal file. If another journal file is queued, **gltj** opens the next file. For example:

```
gltj> close
gltj>
```

3. Exit from **gcon**.

Using Journaling to Recover From a Disaster

If the database you are journaling becomes unusable due to a system disaster (i.e. a disk crash, a network failure) or a human error, you can restore it and all of the committed changes made up to the point the disaster occurred.

The following sections describe how to recover from a system disaster and how to recover from a human error.

Recovering from a System Disaster

If your system crashes, you can restore a database from an after-image file following these steps:

1. After your system has crashed, restart the journal server process, close the current journal file, check that an output file is queued, and resume journaling. For example:

```
% gltj &
[2] 8756

%gcon
gltj> Journal server console program
Journaling disabled; current output file: journ.a
Output files queued:
        journ.b
Known databases and connections:
      /usr/interbase/journal.dir/atlas.gdb (1), connections: 0
gltj> close
gltj> queue journ.c
gltj> resume
gltj>
```

2. Validate the journaled databases by using the **gfix validate** command. If **gfix** does not return any errors, then your databases are safe to use. If **gfix** does return errors, continue with this recovery procedure. For example:

O/S	Command
Apollo AEGIS	% gfix atlas.gdb -validate -full
UNIX	% gfix -validate -full atlas.gdb

3. If you need to recover more than one database, use **qli** to verify the names and order of the journal files.

```
% qli
Welcome to QLI
Query Language Interpreter
QLI> ready journal.gdb
print sequence, date_opened, status, filename of journal_files
```

```
            DATE
SEQUENCE    OPENED          STATUS      FILENAME
========    ============    =========   ============
1           03-Mar-1990     closed      /journal.dir/journ.a
2           03-Mar-1990     active      /journal.dir/journ.b
3           03-Mar-1990     queued      /journal.dir/journ.c

QLI> exit
```

4. Invoke the **grst** utility. **grst** prompts you for the names of the permanent journal files. Enter the names of the journal files you want to recover. Press Control-D when you want to exit from **grst**. Do not enter the name of the permanent journal file activated after the system crash. This file does not contain any journal data, and it is locked against your use.
5. Backup and restore the database created by **grst**. This step is necessary for rebuilding all internal structures. InterBase does not journal changes to indexes. Restoring the database restores the indexes. For example:

O/S	Command
Apollo AEGIS	% gbak atlas.gdb atlas.gbak % gbak atlas.gbak new_atlas.gdb -create
UNIX	% gbak atlas.gdb atlas.gbak % gbak -create atlas.gbak new_atlas.gdb

Recovering from a Human Disaster

If you want to restore your journal files to a specific time and date before a disaster occurred, use the **grst -u** "***date timestamp***" command. For example, suppose a user inadvertently corrupts the database on Friday around 2:00. You can use the **grst -u** "***date timestamp***" command to restore your journal files just prior to the time the corruption occurred.

To restore your journal files to a specific time and date:

1. As soon as the problem is discovered, invoke **gcon** to close the current journal file. For example:

```
% gcon
gltj> close
gltj>
```

2. Invoke the **grst** utility and use the **-u *"date timestamp"*** option to restore your database up to the time just before the disaster occurred. When specifying the

date, use the DD-MMM-YY format. When specifying the time, use the HH:MM:SS format. If you do not specify the time, **grst** defaults to the beginning of the day you specified. Be sure to enclose the date and timestamp in quotes.

When **grst** prompts you for the name of the journal filename, be sure to enter the pathname and not just the name of the journal file. For example:

```
% grst -u "03-Mar-1990 12:10:30"
Enter journal filename: /journal.dir/journ.a
Database "/journal.dir/atlas.gdb" (1) enabled at 15:11:10
03/01/90
Recover it? [y/n]:y

Enter journal filename: /journal.dir/journ.b
Database "/journal.dir/atlas.gdb" (1) enabled at 15:12:10
03/01/90
Recover it? [y/n]:y

Enter journal filename: (Press Control-D to exit.)
%
```

grst continues to process the journal file after the date and timestamp you entered. At each commit prompt, you are asked if you want to apply the transaction to the journal file. If you want to apply the transaction, answer yes. If you do not want to apply the transaction, answer no.

3. Backup and restore the database created by **grst**. This step is necessary for rebuilding all internal structures. InterBase does not journal changes to indexes. Restoring the database restores the indexes.

O/S	Command
Apollo AEGIS	% gbak atlas.gdb atlas.gbak % gbak atlas.gbak new_atlas.gdb -create
UNIX	% gbak atlas.gdb atlas.gbak % gbak -create atlas.gbak new_atlas.gdb

VMS Journaling

The next several sections describe how to start, enable, and disable journaling, and how to restore a database from an after-image journal file on VMS systems.

Customizing the VMS Journaling Environment

Before you begin journaling on VMS, you should decide how you want to customize your journaling environment. You can customize the after-image journaling environment to suit your application. For example, you can choose:

- *One database per journal directory.* By establishing one journal directory per journaled database, you give each database its own set of permanent journal files. The reduces the amount of data written to each journal file, making recovery quicker. However, this option requires more system resources. If you write journal files directly to tape, you must have one tape drive for each journaled database.
- *Multiple databases per journal directory.* By choosing one journal directory for several databases, you can journal change records to a single file. This option uses your system resources more efficiently because it writes directly to tape. However, recovering any one database takes longer, because the journal contains records relevant to other database.
- *Disk journaling.* If you specify a disk file as the permanent journal file, **gltj** writes the permanent journal to disk. This provides better performance than writing to tape, but requires monitoring the size of the journal files so the journal does not overflow a disk. Permanent journal files can be archived to tape using VMS BACKUP or some other backup utility. If you choose to journal to disk, *do not* write your journal file to the same device as your database.
- *Tape journaling.* You can specify a tape file as the permanent journal file. This option requires dedicating a tape drive, or a set of tape drives, to database journaling. It also requires having an operator available to change the tapes as necessary. By establishing a tape volume set, you can automatically transfer from one tape to the next.
- *Cluster journaling.* InterBase after-image journaling works transparently on all nodes on a cluster, although the **gltj** program runs only on one node in a cluster. Applications journal their changes to the intermediate journal file even if **gltj** is not available. Thus, journaling continues in a cluster even if the node running the **gltj** process crashes. However, only **gltj** writes changes from the intermediate journal file to the permanent journal files. If that node is unavailable for an extended period of time, some other node should start the **gltj** process.

Starting VMS Journaling

To set up your journal directory and start **gltj**:

1. Create the journal directory. The following commands create a journal directory on a disk called *$jrndisk*, and makes it the default directory.

```
$ create /dir $jrndisk:[journal]
$ set def $jrndisk:[journal]
```

2. Create a journal database in the journal directory. The InterBase installation procedure leaves a backup copy of the journal database in the *sys$library* directory. The following command uses **gbak** to create a journal.gdb database:

```
$ gbak /create sys$library:journal.gbak journal.gdb
```

3. Run the journal server:

```
$ gltj
> status
18:6:41 03/01/90 Journal server started
Journaling disabled; no output file suspended:
```

   The **gltj** program does not prompt you for input.

4. Queue one or more files. In this example, the files are created on another device called *$jrndev*.

```
> queue $jrndev:journ.a
18:6:59 03/01/90 Journal file "$jrndev:journ.a" opened

> queue $jrndev:journal.b
```

5. Check the status of journaling:

```
> status

Journaling enabled; current output file: $jrndev:journ.a Output
files queued:
  $jrndev:journ.b
```

The following section explains how to enable after-image journaling.

Enabling VMS After-Image Journaling

When you enable journaling for a database, InterBase automatically backs up the database into the journal files, so you don't have to perform a separate backup or locate the right backup when you need to recover the database. Once journaling is enabled for a

database, that database cannot be changed unless the intermediate journal file is accessible, so no intermediate changes can be lost.

To enable VMS after-image journaling:

1. Use **gfix** to enable journaling for the database.

```
$ gfix names.gdb /enable $jrndisk:[journal]
```

2. Check to see if **gltj** reports that journaling has been enabled.

```
$ gltj
18:43:37 03/01/90 Journaling enabled for
"$DISK1:[JONES.WORK]NAMES.GDB;1" (1)
18:43:46 03/01/90 Process sign_off for
"$DISK1:[JONES.WORK]NAMES.GDB;1" (1)
```

Enabling journaling requires exclusive access to the database. If other users are active on the database, the **gfix** command does not complete until the other users detach from the database.

Monitoring Journaling Status

gltj writes a status line to the database each time a process attaches or detaches a database. It also reports each time journaling is enabled or disabled for a database. Each time someone attaches to the database, **gltj** reports the event. For example:

```
18:47:15      03/01/90 Process sign_on for
"$DISK1:[JONES.WORK]NAMES.GDB;1" (1)

18:47:51  03/01/90 Process sign_on for
"$DISK1:[JONES.WORK]NAMES.GDB;1" (1)

19:4:45  03/01/90 Process sign_on for
"$DISK1:[JONES.WORK]NAMES.GDB;1" (1)

19:4:55  03/01/90 Process sign_off for
"$DISK1:[JONES.WORK]NAMES.GDB;1" (1)

19:5:15  03/01/90 Process sign_on for
"$DISK1:[JONES.WORK]NAMES.GDB;1" (1)

19:12:46  03/01/90 Journaling enabled for
"$DISK1:[JONES.WORK]ANOTHER.GDB;1" (2)
```

```
19:12:49  03/01/90 Process sign_off for
"$DISK1:[JONES.WORK]ANOTHER.GDB;1" (2)
```

To get the current status of a database, use the **status** command.

3. Type **status** to get a status report from **gltj**:

```
> status
Journaling enabled; current output file: $jrndev:journ.a

Output files queued:
$jrndev:journ.b
Known databases:
   $DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)
   $DISK1:[JONES.WORK]NAMES.GDB;1 (1)
```

Changing Journal Files

Whether you choose to journal to tape or journal to disk and archive files to tape, occasionally you need to close one permanent journal file and open another. The following **gltj** commands allow you to change journal files, creating reliable checkpoints in the journal.

- **Queue** introduces a new permanent journal file.

```
> queue $jrndev:journ.a

18:6:59  03/01/90 Journal file "$jrndev:journ.a" opened

> queue $jrndev:journal.b
```

- **Flush** flushes all changes from the intermediate journal file to the permanent file and suspends journaling. This is an optional step you can skip if you discover you have run out of space for the current permanent journal file. However, you do not have a reliable checkpoint unless you flush the intermediate file before closing the current permanent journal file.

```
> flush

suspended:
```

- **Close** closes the current permanent journal file. If another file is queued, **gltj** opens the next file.

```
> close

19:39:46  03/01/90 Journal file "$jrndev:journ.b" opened
Journaling suspended; current output file: $journdev:journ.b
```

```
Known databases:
$DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)
$DISK1:[JONES.WORK]NAMES.GDB;1 (1)

> queue $jrndev:journ.c
```

- **Resume** causes **gltj** to resume writing changes from the intermediate journal to the permanent journal file. This is an optional step. If you have not used the **flush** command, the journal server is not suspended, and you do not need to **resume**.

```
suspended:

> resume
Journaling enabled; current output file: $jrndev:journ.b Output
files queued:
         $jrndev:journ.c
Known databases:
      $DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)
      $DISK1:[JONES.WORK]NAMES.GDB;1 (1)
```

If you journal to disk, archiving files to tape with the VMS backup you can queue a series of files on different disks. As one disk fills, close that file, archive it, delete it from disk, and requeue the file.

If you journal to tape, you may need to suspend journaling while you change media without changing the journal file. To suspend journaling without changing files use these **gltj** commands:

- **Suspend** causes **gltj** to cease writing from the intermediate journal. Changes are still written to the intermediate journal, which eventually fills up, stopping all database activity, unless you restart **gltj**.

```
> suspend
Journaling suspended; current output file: $jrndev:journ.b
Output files queued:
$jrndev:journ.c
Known databases:
          $DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)
        $DISK1:[JONES.WORK]NAMES.GDB;1 (1) suspended:
```

- **Resume** causes **gltj** to resume flushing changes from the intermediate journal to the permanent journal file.

```
suspended:

> resume
Journaling enabled; current output file: $jrndev:journ.b
Output files queued:
```

```
          $jrndev:journ.c
Known databases:
          $DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)
         $DISK1:[JONES.WORK]NAMES.GDB;1 (1)
```

Disabling VMS Journaling

If you decide to discontinue journaling for a database, use the **gfix/disable** option. Remember, journaling should not be casually disabled, since reenabling journaling involves backing up the database again. You might disable journaling if you plan a massive batch update of the database and want to restart your journal set (collection of permanent journal files) when the update is done. You should disable and reenable journaling periodically to limit the size of the journal set that InterBase reads to recreate your database.

To disable journaling:

1. Use **gfix** to disable journaling for the database. For example:

   ```
   $ gfix names.gdb /disable
   ```

2. Check the **gltj** log.

   ```
   19:19:35  03/01/90 Journaling disabled for
   "$DISK1:[JONES.WORK]NAMES.GDB;1" (1)
   19:19:36  03/01/90 Process sign_off for
   "$DISK1:[JONES.WORK]NAMES.GDB;1" (1)

   > status

   Journaling enabled; current output file: $jrndev:journ.a Output
   files queued:
     $jrndev:journ.b
   Known databases:
           $DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)
           $DISK1:[JONES.WORK]NAMES.GDB;1 (1)
   ```

Note

A database remains known to the journal server even after being disabled.

Stopping and Restarting Journaling

To stop InterBase after-image journaling, use the **gltj** commands **flush** and **shutdown**. **Flush** empties the intermediate journal file and suspends journaling. **Shutdown** causes the **gltj** program to exit. Restart journaling by reinvoking the journal server from the same journal directory. When you restart, the journal server is suspended. You should flush the journal again to be certain all data has been written to the permanent file, then close the current journal file before resuming journaling. For example, to stop journaling:

1. Use the **flush** command to flush all changes from the intermediate journal file to the permanent file and mark the intermediate file as empty:

```
> flush
suspended:
```

2. Use the **shutdown** command to exit the server:

```
> shutdown
19:46:0  03/01/90 Journal server shutdown $
$
```

3. Restart the journal server. When **gltj** starts, it suspends itself. You should first flush the intermediate journal, then close the current journal file. Closing the current journal file is not necessary, but the point at which a journal server is shut down and restarted is often a convenient checkpoint.

```
$ gltj

19:47:9  03/01/90 Journal server started
19:47:13  03/01/90 Journal file "$jrndev:journ.b" opened
Journaling suspended; current output file: $jrndev:journ.b
Output files queued:
          $jrndev:journ.c
Known databases:
          $DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)
suspended:

> flush
suspended:

>queue $jrndev:journ.d
close

19:47:24  03/01/90 Journal file "$jrndev:journ.c" opened
Journaling suspended; current output file: $jrndev:journ.c
Output files queued:
          $jrndev:journ.d
```

```
Known databases:
          $DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)
suspended:

> resume

Journaling enabled; current output file:
      $jrndev:journ.c
Output files queued:
          $journdev:journ.d
Known databases:
          $DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)

> queue $jrndev:journ.d
```

Recovering from a System Disaster

Sometimes it is not possible to close down the journal server in an orderly fashion. Closing down the journal server with a power failure or a system crash may result in some journal records being written from the intermediate file to the permanent journal twice—once before the crash and once after. The recovery program recognizes this situation and compensates for it.

If you have a system crash, you should first restart **gltj** so any data remaining in the intermediate journal file is flushed to the permanent journal file. Then check your databases using the **validate** option of **gfix**. In most cases, they do not require any further attention. Should one of the databases prove unusable, recover the database with **grst**, the database recovery program. Once **grst** has recreated the data, backup and restore the database with **gbak** to insure all internal structures have been completely recreated.

To recover from a system crash:

1. Following the crash, restart the journal server, flush the intermediate journal file, check that another output file is queued, close the current permanent journal file, queue another file if necessary to keep one in the queue, and resume journaling to avoid blocking general database use. For example:

```
$ gltj

19:47:9  03/01/90 Journal server started
19:47:13  03/01/90 Journal file
$jrndev:journ.b" opened
Journaling suspended; current output file: $jrndev:journ.b
Output files queued:
      $jrndev:journ.c
```

```
Known databases:
          $DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)
suspended:

> close

19:47:24  03/01/90 Journal file "$jrndev:journ.c" opened
Journaling suspended; current output file: $jrndev:journ.c
Known databases:
          $DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)
suspended:

> queue $jrndev:journ.d
> resume

Journaling enabled; current output file: $jrndev:journ.c Output
files queued:
          $jrndev:journ.d
Known databases:
          $DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)
```

2. Validate journaled databases. If the **gfix validate** command does not return any errors, the databases are safe to use. If there are errors, continue with the recovery procedure.

```
$ gfix $disk1:[harrison.work]names.gdb /validate /full
$ gfix $disk1:[harrison.work]another.gdb /validate /full

file $disk1:[harrison.work]another.gdb is not a valid database
```

3. If you need to recover one or more databases, use **qli** to verify the names and order of the journal files. For example:

```
$ qli

Welcome to QLI
Query Language Interpreter
QLI> ready journal.gdb
QLI>  print sequence, date_opened, status, filename of
journal_files

             DATE
SEQUENCE      OPENED        STATUS           FILENAME
=========  ===========  =========  ======================

     1       24-Oct-1988  closed     $jrndev:journ.a
      2      24-Oct-1988  closed     $jrndev:journ.b
```

```
        3       24-Oct-1988  active     $jrndev:journ.c
        4       24-Oct-1988  queued     $jrndev:journ.d

    QLI> exit
```

4. Invoke the recovery program, **grst**. **grst** prompts for journal file names. Each time it encounters a record of journals being enabled for a database, it prompts to see if that database should be recovered, and prompts for a name for the new database. It continues to prompt for additional journal files until you respond with Control-Z. Do not enter the name of the permanent journal file which you activated after the system crash. This file does not contain any data written before the crash, and is locked against your use.

```
$ grst
Enter journal filename: $jrndev:journ.a Database
"$DISK1:[JONES.WORK]NAMES.GDB;1"
(1) enabled at 18:43:35  03/01/90
Recover it? [y/n]: n
Database "$DISK1:[JONES.WORK]ANOTHER.GDB;1"
(2) enabled at 19:12:41  03/01/90

Recover it? [y/n]: y Enter filename for database: foo.gdb Enter
journal filename: $jrndev:journ.b Enter journal filename:  Enter
journal file name to process journal, or <EOF> to finish Enter
journal filename: *EXIT*
```

5. Backup and restore the database created by **grst** to be certain all internal structures are correctly built. InterBase does not journal changes to indexes, so the database created by **grst** does not perform as well as it might otherwise.

```
$ gbak foo.gdb foo.gbak
$ gbak /create foo.gbak yet_another.gdb
```

Recreating a Database up to a Journal Checkpoint

Human error or other conditions might require that you recreate a database from an earlier checkpoint. When you close one journal file and open another, you can create a checkpoint to which you can revert.

The procedure described in the section *Changing Journal Files* explains how to create a reliable checkpoint if you include the **flush** command described in the procedure as Step 2.

The procedure described below shows the use of a check point to restore a database to the state it was in shortly before an error occurred at 7:42 PM, destroying the contents of *another.gdb*.

To recreate a journal up to a journal checkpoint:

1. As soon as the problem is discovered, flush and close the current journal file.

```
> flush

suspended:

> close

19:47:24  03/01/90 Journal file "$jrndev:journ.c" opened
Journaling suspended; current output file: $jrndev:journ.c
Known databases:
          $DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)
suspended:

> queue $jrndev:journ.d
> resume

Journaling enabled; current output file: $jrndev:journ.c Output
files queued:
      $jrndev:journ.d
Known databases:
          $DISK1:[JONES.WORK]ANOTHER.GDB;1 (2)
```

2. The journal database contains the time as well as the date each new journal file was opened. By using an edit string on the **qli** print line, you can display the times.

```
$ qli
Welcome to QLI Query Language Interpreter
 QLI> ready journal.gdb
 QLI> print sequence, date_opened using dd-mmm-
 yyyybtt:tt:tt.tt,  status, filename of journal_files

DATE
SEQUENCE     OPENED                    STATUS  FILENAME=
========     ================          ======  ===========

       1     24-Feb-1990 18:06:59.00   closed  $jrndev:journ.a
       2     24-Feb-1990 19:39:46.00   closed  $jrndev:journ.b
       3     24-Feb-1990 19:47:24.00   active  $jrndev:journ.c
       4     24-Feb-1990               queued  $jrndev:journ.d
```

3. Invoke **grst**. **grst** prompts for journal file names. Start with the first journal file. Each time **grst** encounters a record of journal being enabled for a database, it prompts you to specify if that database should be recovered, and prompts for a name for the new database. In the following example, the database *another.gdb* is recovered. Since only the first journal file was closed before the flawed program ran, stop recovery after reading that file.

For More Information

For detailed descriptions of the available options for each of the following utilities, see the entries in Chapter 6, *Database Operations Reference* for:

- **gcon**
- **gfix**
- **gltj**
- **grst**

Chapter 5
Disk Shadowing

This chapter describes disk shadowing.

Overview

InterBase lets you recover from a hardware failure using disk shadowing. A disk *shadow* is a physical copy of a database stored in the same format as a database. A disk shadow consisting of more than one file is called a *shadow set*. Once enabled, a shadow maintains a duplicate copy of a database that is always in sync with the database it is shadowing. If the shadowed database is lost as the result of a hardware failure, simply activate the shadow and begin using it as a database.

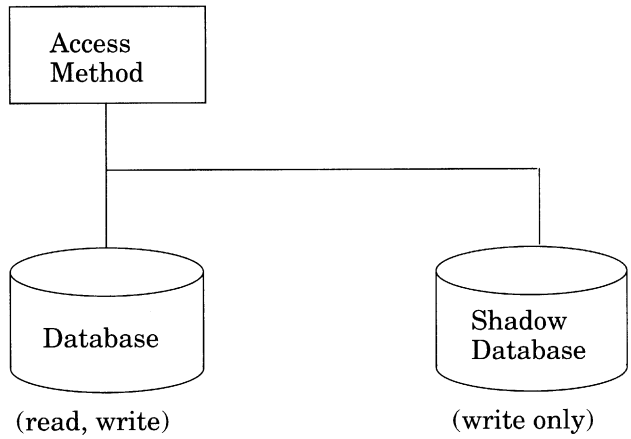
Components of Disk Shadowing

Disk shadowing has the following components:

- RDB$FILES system relation. This system relation lists the shadow files for a database.
- A shadow. A shadow is an InterBase database that contains the same information as the database it is shadowing. You can define multiple shadows for a database.
- A database. This is the database that you want to shadow.

Figure 5-1 diagrams the components of disk shadowing.

Figure 5-1. Disk Shadowing Diagram



Using Disk Shadowing for Disaster Recovery

InterBase provides disk shadowing as another mechanism for disaster recovery. Shadowing allows you to recover from hardware failures, which result in the loss of the main database file. Shadowing does *not* allow you to recover from database corruption caused by users. Compared to after-image journaling as a form of disaster recovery, disk shadowing also has advantages and disadvantages as a means for disaster recovery.

Some of the advantages of disk shadowing are:

- Minimal impact on performance. Only physical write operations are affected by a disk shadow. The main database and the shadow file should be located on two different disks. This allows disk I/O to happen in parallel.
- Quick recovery. A shadow can be instantly activated for immediate use.
- Limited disk usage. A shadow is the same size as the database it is shadowing. If you are shadowing a large database, you can define multiple-file shadows and spread them over several disks.
- Starting a shadow does not require exclusive access to a database. You can start a shadow at anytime, even when the database is in use.
- The shadow file is written by the database process and it does not require a separate process from the database it is shadowing.
- An operator is not required to maintain a shadow.

Shadowing has a few disadvantages. You cannot:

- Shadow to tape.
- Recover the shadow to a specific time.
- Recover from user errors made to the database resulting in database corruption. For example, if someone deletes important data from the database, it is also deleted from the shadow.
- Shadow to a remote location. The shadow must be writable from the machine that owns the disk that owns the database. For example, the shadow must be writable from within a cluster in VMS, within a ring in an Apollo network, and within an NFS mounted file system in a UNIX network.

In addition, shadowing duplicates every physical disk write and requires twice as much available disk space as a database that does not have a shadow.

The following sections describe how to define a shadow, delete a shadow, and activate a shadow.

Defining a Shadow

You can define a shadow two different ways—using the **define shadow** statement in **gdef**, or by directly storing a record in the RDB$FILES system relation in **qli**. Using the **define shadow** statement is the preferred method.

When you define a shadow using the **define shadow** statement, you can specify the pathname, additional files, and a file length, or a starting page. You also specify whether a shadow is automatically or manually deleted if it becomes unavailable. For complete information on the syntax for the **define shadow** statement, see the entry in the *DDL Reference*.

> When you define a shadow for a database, it must be considered a local file. You cannot define a shadow to a remote location. For example, a shadow must be writable within a cluster on VMS, writable within a ring in an Apollo network, or writable within an NFS file system in a UNIX network.

Gdef automatically writes this information to the RDB$FILES system relation. Table 5-1 lists the fields used in the RDB$FILES system relation for defining a shadow .

Table 5-1. RDB$FILES System Relation

Field_Name	Datatype	Length	Description
RDB$FILE_NAME	Text	125	Contains a filename or a valid system pathname for a shadow or a secondary file.
RDB$FILE_SEQUENCE	Short		Specifies the order of files within a shadow set.
RDB$FILE_START	Long		Specifies a starting page number for shadows spanning multiple files. If the starting number is 0 or null, one shadow file is created.
RDB$FILE_LENGTH	Long		Specifies the file length in blocks

Table 5-1. RDB$FILES System Relation continued

Field_Name	Datatype	Length	Description
RDB$SHADOW_ NUMBER	Short		Specifies the shadow set number. This indicates which file the shadow set belongs to. If the value of this field is 0 or missing, InterBase assumes the file being defined is a secondary file, not a shadow file.

Note

For a complete description of all fields in the RDB$FILES system relation, see the appendix on system relations in the *Data Defition Guide*.

To define a shadow:

1. Invoke **gdef** and ready the database for which you want to define a shadow. For example:

```
GDEF> modify database "/target_node/atlas.gdb";
```

2. Define the shadow. Do not define the shadow on the same disk as the main database. The example below defines a single-file shadow for the *atlas.gdb* database. The shadow file is automatically deleted if it becomes unavailable:

```
GDEF> define shadow 1 auto "/target_node/atlas_shadow";
```

When you define a shadow it is an exact copy of the main database.

3. Exit from **gdef**.

```
GDEF> exit
%
```

When you exit from **gdef** and the transaction commits, the shadow is enabled and immediately begins recording updates to the main database. **Gdef** automatically updates the RDB$FILES system relation with the shadow information.

4. Verify that you created the shadow. Invoke **qli**, and ready the database you created the shadow for. Then use the **show database** statement to verify the shadow is active. This example verifies that *atlas_shadow* was created for the atlas.gdb database:

```
% qli
Welcome to QLI
Query Language Interpreter
```

```
QLI> ready atlas.gdb
QLI> show database
Database description:
   ⇓
   Shadow 1, File: /target_node/atlas_shadow starting at page 0
```

5. Exit from **qli**.

```
QLI> exit
%
```

Caution

If you copy a database that has a shadow, both versions of the database update the shadow file.

Defining a Multi-File Shadow

You define a multi-file shadow using the **define shadow** statement in **gdef**. When defining a multi-file shadow, you must specify a starting page number for each file within the shadow.

To define a multi-file shadow:

1. Invoke **gdef** and ready the database for which you want to define the shadow. For example:

```
GDEF> modify database "/target_node/atlas.gdb";
```

2. Define shadow. The example below defines a two-file shadow with each file starting at a different page:

```
GDEF> define shadow 2 auto "/target_node/atlas_shadow_a"
starting CON> at page 100
CON> file "/target_node/atlas_shadow_b" starting at page 200
CON> file "/target_node/atlas_shadow_c" starting at page 300;
```

3. Exit from **gdef**.

```
GDEF> exit
%
```

When you exit from **gdef** and the transaction commits, the multi-file shadow is enabled and immediately begins recording updates to the main database. **Gdef** automatically updates the RDB$FILES system relation with the shadow information.

4. Verify you created the shadow. Invoke **qli** and ready the database you created the multi-file shadow for. Then use the **show database** statement to verify the shad-

ow files are active. The example verifies that all of the shadow files for the atlas.-gdb database are active:

```
% qli
Welcome to QLI
Query Language Interpreter
QLI> ready atlas.gdb
QLI> show database
Database description:
  ⇓
Shadow 2, File:/target_node/atlas_shadow_a starting at page 0
Shadow 2, File:/target_node/atlas_shadow_b starting at page 101
Shadow 2, File:/target_node/atlas_shadow_c starting at page 201
```

5. Exit from **qli**.

```
QLI> exit
%
```

Adding an Additional File to a Shadow

If you think your database might grow to be quite large, you might decide to add additional files to your existing shadow for the database. Adding a file to an existing shadow is like adding a file to an existing database. If you add additional files to your shadow, as your database grows and the first shadow file fills up, updates to the database automatically overflow into the next shadow file.

Note

Adding a file to a shadow requires exclusive access to the database.

When you add an additional file to a shadow, you can assign the file any page size value. If the value you assign is less than the size of the database, the value increments one greater than the size of the current database. If the value you assign is greater than the database size, a stub file is created. When the database reaches the assigned value, it overflows into the secondary file.

To add an additional file to a shadow:

1. Invoke **gdef** and ready the database for which you want to add an additional shadow file to. For example:

```
GDEF> modify database "/target_node/atlas.gdb";
```

2. Define the additional shadow file. The example below adds another file to *atlas_shadow*:

```
GDEF> define shadow 1 "/target_node/atlas_shadow_d"
```

3. Exit from **gdef**.

```
GDEF> exit
%
```

When you exit from **gdef** and the transaction commits, the additional shadow file is enabled. Depending on the value you assigned the file it may or may not immediately begin recording updates to the main database. **Gdef** automatically updates the RDB$FILES system relation with the shadow information.

4. Verify that you added the file or files to your shadow. Invoke **qli** and ready the database you added the additional shadow file to. Then use the **show database** statement to verify the shadow files are active. The example verifies the existence of all shadow files for the atlas.gdb database:

```
% qli
Welcome to QLI
Query Language Interpreter
QLI> ready atlas.gdb
QLI> show database
Database description:
   ⇓
 Shadow 1, File: /target_node/atlas_shadow starting at page 0
 Shadow 1, File: /target_node/atlas_shadow_c starting at page
101
```

5. Exit from **qli**.

Deleting a Shadow

If you no longer want to shadow a database, you can delete the physical shadow files using the **delete shadow** statement in **gdef**. Once you delete a shadow, it immediately stops recording updates from the database.

To delete a shadow:

1. Invoke **gdef** and ready the database that has the shadow you want to delete. For example:

```
GDEF> modify database "/target_node/atlas.gdb";
```

2. Delete the shadow. The example below deletes the atlas_shadow:

```
GDEF> delete shadow 1
```

3. Exit from **gdef**.

```
GDEF> exit
```

When you exit from **gdef** and the transaction commits, the shadow stops recording updates from the main database. In addition, the shadow is automatically deleted from the RDB$FILES system relation.

4. Verify you deleted the shadow. Invoke **qli** and ready the database you created the shadow for. Then use the **show database** statement to verify the shadow is no longer active. For example:

```
% qli
Welcome to QLI
Query Language Interpreter
QLI> ready atlas.gdb
QLI> show database
Database description:
  ⇓
The information about the shadow file does not display in  your
window or on your terminal.
```

5. Exit from **qli**.

```
QLI> exit
```

Transaction Considerations

When you delete a shadow that is recording updates from the main database, it is important to understand what happens to transactions during the time you are deleting the shadow. The following list describes the possible outcomes of the update during the transaction to delete the shadow:

- Updates to the main database occurring within the transaction that deleted the shadow are recorded in the shadow.
- Updates to the main database occurring after the shadow is deleted are not recorded in the shadow.
- Active transactions in the main database are considered rolled back in the shadow when you delete the shadow.

Activating a Shadow

If the database you are shadowing becomes unavailable due to a hardware failure, you can immediately activate your shadow using the **gfix activate** command. *Be sure the shadow is not in use before you activate it.* A shadow is active if the main database has active transactions. To activate a shadow, specify the pathname of the first file in a shadow.

Caution

Do not activate a shadow unless your main database becomes corrupt.

The following command activates a shadow:

O/S	Command
Apollo AEGIS	% gfix //jeeves/dbname_shadow -a
UNIX	% gfix -a wanda:/dbname_shadow
VMS	$ gfix/activate disk1:[recover]dbname_shadow

Once you activate the shadow, you may want to change the name of the shadow to the name of your original database.

Troubleshooting

The following section describes the errors you might encounter when you activate or define a shadow. It also suggests a solution for the problem:

- shadow accessed; cannot attach active shadow file

  You tried to attach to a database currently attached to another database as a shadow. If the original database is available, erase the records in the RDB$FILES system relation that define the shadow file. Otherwise, use the **gfix activate** command to convert the shadow to an active database.

- shadow missing; a file in the "pathname" is unavailable, shadow deleted

  A file in the shadow has been deleted or is inaccessible. The shadow entry is deleted from the RDB$FILES system relation in order to preserve the integrity of the database. Redefine the shadow.

For a complete list of **gdef** error messages that can occur when defining additional files, see the chapter on defining databases in the *Data Definition Guide*.

For More Information

For more information on:

- Alternative disaster recovery mechanisms, see Chapter 4, *Journaling.*
- Syntax for the **define shadow** and **delete shadow** statements, see *DDL Reference.*
- RDB$FILES system relation, see the appendix on system relations in the *Data Definition Guide.*
- **gdef**, see the *Data Definition Guide.*

Chapter 6
Database Operations Reference

This chapter provides reference information on the InterBase utilities.

Overview

The following InterBase utilities are described in this chapter:

- **gbak**
- **gcon**
- **gscu**
- **gfix**
- **grst**
- **gltj**

Gbak

Function

The **gbak** utility backs up and restores databases.

Syntax

```
gbak [options] file1 file2 [integer[,file3
[integer]...]
```

Options

b[ackup_database]
Backs up *file1* to *file2*. This is the default database. You must provide **gbak** with the secondary file specification in order to restore the database to multiple files. Use this option when you backup a database.

c[reate_database]
Restores a database from a backup file *file1* to a new file *file2*. **Create_database** has the following operating system restrictions:

- VMS creates a new version of the database file
- UNIX and Apollo systems fail because **gbak** overwrites existing files only if you specify the **replace** switch.

Use this option when you restore a database.

d[ev] {mt|ct}
Specifies the device name when you backup a database to tape or restore from tape. The device name is **mt** for magnetic tape or **ct** for cartridge tape. This is an option for Apollo machines only. Use this option when you backup a database.

e[xpand]
Tells **gbak** to produce an expanded database for a restore. Use this option if you are restoring the backup with a pre-Version 2.4 version of **gbak**. Use this option when you backup a database.

f
Restores database files starting at a specified page number. When you restore a multi-file database, **gbak** lists the multiple output files and their page size. The **f** option is optional. Use this option when you restore a database.

g
Enables you to run **gbak** without performing garbage collection during database backup. **Gbak** only sweeps the active copy of the database and not the previous versions when the **-g** switch is used. This prevents the sweeping of old records.

i[**nactive**]
Makes indexes inactive so you can back up a database with duplicate unique index values. **Gbak** does not work on databases with duplicate unique index values unless you use this option. Use this option when you restore a database.

-k(ill)
Restores database without its shadow.

l[**imbo**]
Ignores transactions that are in limbo. Use this option when you backup a database.

m[**etadata**]
Backs up only the data definition in *file1* to *file2*. Specify this option only if you want an empty version of an existing database. Use this option when you backup a database.

n[**o_validity**]
Does not restore database validity conditions. Use this option when you restore a database.

o[**ne_at_a_time**]
Commits after each relation in a database is restored. This option is best used with the **verify** option. Use this option if you are having problems restoring a database. It can help determine if a particular relation is a source of trouble. Use this option when you restore a database.

p[**age_size**] *integer*
Changes the page size of an existing database. You must first back up the database without specifying this option, and then restore it with the option. Use this option when you restore a database.

Integer specifies the new page size. The default page size for a database is 1024 bytes. You can specify a page size of 1024, 2048, 4096, or 8192 bytes. The advantage of a large page size is that it allows a more shallow "tree" structure in the index. Each index bucket is one page long, so longer pages mean larger

buckets and fewer levels in the hierarchy. Use this option when you restore a database.

r[eplace_database]
Restores a database from a backup file *file1*, replacing any existing file with the same name as *file2*. If you do not want to overwrite the existing file, use the **create_database** option instead of this one. Use this option when you restore a database.

t[ransportable]
Builds a transportable backup. Long numeric datatypes such as quads or dates are converted to a generic form. For example, by using the **t** option, you can **gbak** a database on Apollo and restore on VMS. Use this option when you backup a database.

v[erify]
Provides detailed information about what is happening, as it happens. Use this option when you backup a database.

y
Supresses output of error messages. Not available in V3.0.

z
Displays the version number of the software of this utility and the access method.

file1
file2 [integer]
file3 [integer]
Valid file specifications. The database you access may be on another computer in the network. This is called a remote database, and the computer where it is stored on is called the *remote node*. The node you are using is called the *local node*. If the database you access is on the same node as you are, then it is called a *local database*. To access a remote database, use the full network pathname of the database file or establish a logical link to it.

If you are in a directory other than the one that contains the database file, the *filespec* must include the pathname. If the database is on another node, the *filespec* must include the node name and pathname. You can define a link or logical name for the database file.

File specifications for remote databases have the following form:

From	To	Syntax
VMS	VMS via DECnet	node-name::filespec
VMS	ULTRIX via DECnet	node-name::filespec
VMS	non-VMS and non-ULTRIX	node-name^filespec
ULTRIX	VMS via DECnet	node-name::filespec
Apollo	Apollo	//node-name/filespec
Everything Else	Whatever is left	node-name:filespec

When you restore a database, you can place it in several files using an optional file-length list. The first file is the *primary* file, and subsequent files are called *secondary* files. The primary file is the first database file that is used by the InterBase access method, and secondary files are where database pages are allocated after the primary file is filled. You must specify a length in database pages for each secondary file.

You will probably want to put secondary files on separate disks because their purpose is to allow databases to grow beyond the limits of a single disk. However, you must ensure that all files in a database can be accessed directly by one program running on some computer:

- On VMS systems, all files must be on disks mounted by the same host or cluster-wide devices available to that host.
- On Apollo systems, the files must be in the same ring.
- On Ultrix and Sun systems, the files must be in the same directory tree, and cannot be mounted with NFS.

Examples

The following command backs up a database:

O/S	Command
Apollo AEGIS	% gbak atlas.gdb atlas.gbak -b

O/S	Command
UNIX	% gbak -b atlas.gdb atlas.gbak
VMS	$ gbak/backup atlas.gdb atlas.gbak

The following command backs up a remote database. If you are running on a UNIX system, you must enclose the remote database name in single quotes:

O/S	Command
Apollo AEGIS	% gbak hyde:$disk1:[davis.work]atlas.gdb atlas.gbak -b
UNIX	% gbak -b 'hyde:$disk1:[davis.work]atlas.gdb' atlas.gbak
VMS	$ gbak/b/v gordon^/davis/work/atlas.gdb atlas.gbak

The following example restores a backed up database to a new file:

O/S	Command
Apollo AEGIS	% gbak atlas.gbak new_atlas.gdb -c
UNIX	% gbak -c atlas.gbak new_atlas.gdb
VMS	$ gbak/create_database atlas.gbak-new_atlas.gdb

The following example builds a transportable backup database file:

O/S	Command
Apollo AEGIS	% gbak atlas.gdb atlas.gbak -t
UNIX	% gbak -t atlas.gdb atlas.gbak
VMS	$ gbak/transportable atlas.gdb atlas.gbak

The following example backs up a database on an Apollo system to a magnetic tape:

O/S	Command
Apollo AEGIS	% gbak atlas.gdb -b -dev mt

The following example restores a backed up database and replaces an existing file:

O/S	Command
Apollo AEGIS	% gbak atlas.gbak atlas.gdb -r
UNIX	% gbak -r atlas.gbak atlas.gdb
VMS	$ gbak/replace_database atlas.gbak atlas.gdb

The following example restores a backed up single-file database to a multiple-file database:

O/S	Command
Apollo AEGIS	% gbak bigdatabase.gbak big_db.gdb 10000, big_db.gdba 10000, big_db.gdbb 10000 -r
UNIX	% gbak -r bigdatabase.gbak big_db.gdb 10000, gig_db.gdba 10000, big_db.gdbb 10000
VMS	$ gbak/replace_database bigdatabase.gbak - big_db.gdb 10000, big_db.gdba 10000, big_dbgdbb 10000

Troubleshooting

You may encounter the following messages when you use **gbak**:

- *page size specified <n> greater than limit (8192) bytes*
  page size specified <n> rounded up to <n> bytes
  page size parameter is missing

  When you use the **page_size** option, you must specify an increment of 1024, up to a maximum of 8192. If you specify a number that is not an increment of 1024, InterBase rounds up to the nearest increment.

- *page size is allowed only on restore or create*

  You used the **page_size** option for a backup. You can use this option only during a restore.

- *redirect location for output is not specified*

  You specified an option reserved for future use by InterBase.

- *unknown switch <switch>* or *found unknown switch*

  You chose an option that is unknown or unsupported. See the list above for supported options.

- *conflicting switches for backup / restore*

  At least two options you chose conflict with each other. For example, you tried to backup and restore simultaneously.

- *requires input and output filename*

  You did not specify an input or an output filename. Correct the command and try again.

- *input and output have the same name. Disallowed.*

  You gave the input and output file the same name. Correct the command and try again.

- *REPLACE specified, but the first file <file> is a database*

  You tried to restore a database, but specified a database file as the backup file. Be sure the first file you specify is a backup file, correct the command, and try again.

- *database <database-name> already exists. To replace it, use the -r switch*

  Gbak does not overwrite an existing database when you use the **create** option. Change the name of the database or use the **replace** option to overwrite the existing database.

- *can't open backup file <file1>, <file2>*

  For some reason, **gbak** cannot open the backup file. Check the filenames and the files themselves, correct the command, and try again.

- *database <file> already exists. To replace it, use the -r switch.*

  You tried to overwrite a database file on a UNIX or Apollo system without specifying the **replace** option. Correct the command and try again.

- *don't recognize <switch> attribute <attribute> -- continuing*

  You specified an unknown parameter on the listed option, but **gbak** continues processing the command.

- *unexpected end of file on backup file*

  Gbak encountered the end of the backup file before it expected it. Be sure the backup file does not terminate abnormally. If it does, call Interbase Customer Support at 1-800-437-7367.

- *string truncated*

  You may encounter the following message when you use **gbak** on an Apollo:

- multiple sources or destinations specified

  When you used the **device** option, you specified two filenames. Supply one filename and one device.

- *device type not specified* or *device type <device> not known*

  When you used the **device** option, you did not specify a device after the option, or you specified a device incorrectly.

- *gbak" - name not found (OS / naming server)*

  This means that **gbak** does not show up anywhere along your shell's command search list. Check the search path, correct it if necessary, and try again.

- *can't close APOLLO tape descriptor file <filename>*
- *can't create APOLLO cartridge descriptor file <filename>*
- *can't create APOLLO tape descriptor file <filename>*
- *can't set APOLLO tape descriptor attribute for <filename>*
- *input filename will be ignored*
- *output filename will be ignored*

  These are Apollo-related problems. Contact Apollo Customer Support if you receive any of these messages.

See Also

Chapter 2, *Database Backup and Recovery*.

The *Data Definition* guide for more information about multiple-file databases.

Gcon

Function

The **gcon** utility is a console program that communicates with the journal server. It is available only on Apollo and UNIX systems.

Syntax

```
gbak [options] file1 file2 [integer[,file3
[integer]...]
```

Options

q[ueue]
Queues a file or device for journaling to disk. Never journal to the same disk with the database.

c[lose]
Closes the current journal file or device.

s[tatus]
Displays the status of the journal server. Status information includes the state of journaling, the name of the output file, and known databases and connections. This information is stored in the journal database.

sh[utdown]
Shuts down the connection between **gcon** and the journal server.

su[spend]
Suspends journaling temporarily.

r[esume]
Resumes journaling that has been suspended.

h[elp]
Displays options with a brief description.

v[ersion]
Displays the version number of the software.

e[xit]
Exits from the journal server console program.

Examples

The following example shows how to queue a journal file:

```
% gcon
gltj> queue journ.1
```

The following example shows stopping the journal server process:

```
% gcon
gltj> shutdown
```

Troubleshooting

You may encounter the following messages when you use **gcon**:

- *"gcon" - name not found (OS / naming server)*

  This means that **gcon** does not show up anywhere path or shell's command search list. Check your path, correct it if necessary, and try again.

You may encounter the following messages when you use **gcon** on an Apollo:

- *Error occurred during mbx_$get_rec*

  An error occurred reading a response to **gcon** from the **gltj** process.

  This could be caused by any of the following reasons:

  — Network trouble. See your system administrator for help.

  — Incompatible versions of journaling on your system. Check your version of Interbase. For more information on running two versions of InterBase on your system, see the *InterBase Version 3.0 Release Notes*.

  — An mbx server is not running on all of the nodes used for journaling. Check to see if there is an mbx server running all nodes that are using journaling. If a server is not running, start one.

- *Error occurred during mbx_$put_rec*

  An error occurred sending a command to the **gltj** process.

- *Error occurred during mbx_$open*

  An error occurred opening a mailbox connection.

  This could be caused by any of the following reasons:

  — Network trouble. See your system administrator for help.

  — Incompatible versions of journaling on your system. Check your version of Interbase. For more information on

running two versions of InterBase on your system, see the *InterBase Version 3.0 Release Notes*.

— Your **gltj** process or your entire network has gone away. Type **pst** to see if you have a **gltj** process running. If you do not have a **gltj** process running, start one.

You may encounter the following messages when you use **gcon** on any UNIX system:

- *Error occurred during close socket*
  Error occurred during socket read
  Error occurred during socket
  Error occurred during write socket

  These could be caused by any of the following reasons:

  — Network trouble. See your system administrator for help.

  — Incompatible versions of journaling on your system. Check your version of Interbase. For more information on running two versions of InterBase on your system, see the *InterBase Version 3.0 Release Notes*.

  — Your **gltj** process or your entire network has gone away. Check to see if you have a **gltj** process running.
  On:

  - System V UNIX, type **ps -e**

  - UNIX, type **ps -ax**

  - Apollo, type **pst**

  If you do not have a **gltj** process running, start one.

  If you get "*Error occurred during socket*" it may be because your **gltj** process has reached its limit on the number of file descriptors it can open.

  If you receive any of the above messages, call InterBase Customer Support at 1-800-437-7367.

- *Error occurred during journal socket file open*

  You get this error message only if you get one of the following error messages:

  — *Error occurred during journal socket file format*

  You have reached a journal file whose format InterBase journaling does not recognize. Get out of **gltj** and check your journaling directory path.

  — *Error occurred during address version*

Gltj recognizes the version number of the journal file you have opened, but the rest of the file's format is not what **gltj** expects. There may be a problem with the journal file. Get out of **gltj** and check your journaling directory path.

If you receive any of the above three error messages, call InterBase Customer Support at 1-800-437-7367.

See Also

See the entry in this chapter for:

- **gltj**

Chapter 4, *Journaling*.

Gcsu

Function

The **gcsu** utility manages central servers. It also displays system management information about central servers.

Syntax

```
gscu [-d|-r|-k {-a|filespec}]
     [-e server_name filespec]
     [-l]
     [-z]
```

Options

-d
Disables a database by removing it from the list of databases being serviced by a central server. Once a database is disabled, no additional attachments for it are accepted by a central server. However, existing attachments to the database are unaffected by the disable command. A central server terminates if all databases being serviced by it are disabled and no user attachments remain.

-r
Re-enables a disabled database that has active attachments through the central server. You can only re-enable a database when at least one active attachment has existed since the database was disabled. Re-enabling a database permits additional attachments to it through the central server.

-k
Disables a database and detaches all current attachments to the database through a central server. All active transactions in the database are rolled back before the database is detached. A central server terminates if all databases being serviced by it are disabled or killed and no user attachments remain.

-a
Indicates that a disable, re-enable, or a kill command should apply to all databases being serviced by all central servers.

filespec
An explicit pathname and database name. You can specify any number of databases on the command line.

-e
Adds a database to the list of databases being serviced by a specific central server.

server_name
A name you specified as the name of a central server.

-l
Lists the names of all databases being serviced by each central server. For each database, it displays current number of attachments and indicates whether a disable or kill command has been issued for the database.

-z
Displays the version number of the software.

Examples

The following example enables a central server and lists the central servers and the databases being serviced by them. The command syntax for UNIX and VMS is the same.

O/S	Command
UNIX	% gcsu -e sample //jeeves/interbase/atlas.gdb -l
VMS	$ gcsu -e sample $disk1:[myown]atlas.gdb -l

Sample UNIX output for the above command is shown below:

```
Server
        Attaches Flags Database Name

SAMPLE
                0       //jeeves/interbase/atlas.gdb
                3       //jeeves/interbase/emp.gdb
```

The following example disables a database from a central server:

O/S	Command
UNIX	% gcsu -d //jeeves/interbase/atlas.gdb
VMS	$ gcsu -d $disk1:[myown]atlas.gdb

The following example re-enables a database to a central server and lists the central servers and the databases being serviced by them:

O/S	Command
UNIX	% gcsu -r //jeeves/interbase/atlas.gdb -l
VMS	$ gcsu -r $disk1:[myown]atlas.gdb -l

Sample UNIX output for the above command is shown below:

```
Server
        Attaches Flags Database Name

SAMPLE

                2       //jeeves/interbase/atlas.gdb
```

Troubleshooting

You may encounter the following errors when you use the **gcsu** utility:

- *It's likely that no central servers are operating.*

  You tried to invoke the gcsu utility and there are no central servers running.

- No central servers are operating.

  You tried to list the central servers and there are not any running.

- *Database "filespec" could not be found so it was not used.*

  The database you specified is not being serviced by a central server. Use the **gcsu -l** command to list the databases using a central server.

- *Server "server_name" could not be found and so database "filespec" was not enabled.*

  The central server you specified does not exist. Use the **gcsu -l** command to get a listing of the available central servers and use one of those or start a new central server.

- The -e switch may only be used in conjunction with the -l and -z switches.

  You used the **e** option with one or more of the **d**, **k**, or **r** options. Use the **e** option alone or with the **l** or **z** options.

- No command indicated with the -a switch. It will be ignored.

  Specify one or more of the **-d**, **-k**, or **-r** options before specifying the **-a** option.

- No command indicated for "filespec". The name will be ignored.

  You specified a database without specifying a command. You have to associate a **d**, **k**, **r**, or **e** options with a database name.

- Server could not enable database "filespec" because: "gds_$print_status."

  This error prints the contents of the **gds_$print_status**. A central server is started with the **-a** option, and the database attachment fails. Determine what to do based on the message returned by the **gds_$print_status**.

See Also

The installation notes for system specific information on starting a central server.

Gfix

Function

The **gfix** utility performs a variety of database maintenance operations.

Syntax

```
gfix filespec options
```

Options

filespec
Specifies the database file on which to perform one or more of the functions described below.

s[**weep**]
Walks through all records in all relations, releasing space held by records that were rolled back and by out-of-date record versions. You can sweep a database without taking it off line. You might consider running this utility in background at a low priority.

h[**ousekeeping**] *integer*
Sets the automatic garbage collection interval. The integer must be a single non-negative. The default it 20,000 transactions. Setting an interval of 0 disables this option.

-k(ill)
Removes all unavailable shadows from a database.

v[**alidate**]
Walks through physical structures. This option requires exclusive access to the database. By default, it locates and releases pages that are allocated but not assigned to any structure (relations, indexes, and so on). It also reports any corrupt structures.

Options for **validate** are:

- **f**[**ull**] walks through record structures in addition to page structures, releasing record fragments that are not assigned to current records.
- **n**[**o_update**] validates and reports corrupt and misallocated structures, but does not change them.

- **m**[**end**] fixes problems that cause records to be corrupt. By default, mend marks records as corrupt so InterBase skips them.

Most errors that **validate** finds occur if a program aborts, but do not affect the integrity of the database. No committed data is lost, and uncommitted updates are rolled back. However, InterBase may have assigned a page from free space for the records without using it. The validate option reports such pages as "orphans" and assigns them to free space.

The **validate** option also locates problems caused by write errors in the operating system or hardware. These errors may make committed data unrecoverable and cannot be corrected with **validate**. Use the **mend full** options to correct such errors.

If **validate** finds corrupted records, copy the broken database and send the copy to the following address for analysis:

Borland International Inc.
InterBase Support
1800 Green Hills Road
P. O. 660001
Scotts Valley, CA 95067

You can also call Customer Support at 800-437-7367 from anywhere in the United States and Canada or at 408-431-5400 from outside of the United States. In addition, you can fax a description of the problem to 408-439-7808.

Meanwhile, use **mend** to mark bad records. Run **gbak** with the **validate** option again to ensure all is well. Unload and restore the database for your own use.

`two_phase`
Commits or rolls back a transaction, depending on its state. Prompts you for advice if all transactions are prepared and you did not specify a commit or a rollback. Use the **two-phase** option with **list** and **prompt** options.

`l[ist]`
Lists the id numbers of transactions in limbo. Also list the partner, current state, and action automated transaction would

take for multi-database transaction. Use with the **two-phase** and **prompt** options.

p[rompt]
Lists the transactions in limbo, and prompts the user for advice on how to complete the two-phase commit process. Use with the **two-phase** and **list** options.

-x
Sets debug on (for Apollo only).

-z
Prints software version number.

c[ommit] *trans-id* **/ all**
Commits the transaction identified by the transaction id or all transactions in limbo. Also commits the partners of the transactions.

r[ollback] *trans-id* **/ all**
Rolls back the transaction identified by the transaction id or all transactions in limbo. Also rolls back the partners of the transactions.

e[nable] *journal-directory*
Enables after-image journaling to the named journal directory. You must first define a journal server using **gltj**. See the entry in this chapter for **gltj**.

d[isable]
Disables after-image journaling.

a[ctivate]
Activates a shadow file or shadow set. Once you activate the shadow file or shadow set, you can attach to it and begin using it as your database.

i[gnore]
Ignores the checksum errors.

Examples

The following command sets the automatic garbage collection interval to 10,000 transactions:

O/S	Command
Apollo	$ gfix atlas.gdb -h 10,000

O/S	Command
UNIX	$ gfix -h 10,000 atlas.gdb
VMS	$ gfix/housekeeping 10,000 atlas.gdb

The following command locates and releases pages and record structures that were allocated but not assigned:

O/S	Command
Apollo	$ gfix atlas.gdb -v -f
UNIX	$ gfix -v -f atlas.gdb
VMS	$ gfix/validate/full atlas.gdb

The following command lists transactions in limbo, and prompts for advice on how to complete the two-phase commit process:

O/S	Command
Apollo	$ gfix atlas.gdb -t -p -l
UNIX	$ gfix -t -p -l atlas.gdb
VMS	$ gfix/two_phase/prompt/list atlas.gdb

The following command enables after-image journaling on a remote database. If you are running on a UNIX system, you must enclose the remote database name in single quotes:

O/S	Command
Apollo AEGIS	% gfix hyde:$disk1:[davis]atlas.gdb -e
UNIX	% gfix -e 'hyde:$disk1:[davis]atlas.gdb'
VMS	$ gfix/enable gordon^/davis/atlas.gdb

Troubleshooting

You may encounter the following messages when you use **gfix**:

- *database file name <file> already given*

  You included too many command line arguments without leading hyphens.

- *invalid switch*

  An item is not a recognized option.

- *incompatible switch combinations*

  You specified at least two incompatible options on the same command line (for example, **enable** and **disable**), or you specified an option which has no meaning without another option (for example, **prompt** used without **list**).

- journal file pathname required

  The **enable** option requires a journal directory pathname.

- number of transactions per sweep required
  numeric value required
  positive number value required

  The **housekeeping** option requires you provide a single non-negative numeric.

- *transaction number or "all" required*

  The **rollback** and **commit** options require another argument specifying which transactions to alter.

- *please retry, giving a database name*

  You must specify a database name.

- *please retry, specifying an option*

  You must specify some option.

- *More limbo transactions than fit. Try again.*

  The database contains more limbo transactions than **gfix** can print in a single session. Commit or rollback some of the limbo transactions using the **rollback**, **commit**, and **two_phase** options, or the **list** and **prompt** combination, and then try again.

You may encounter the following message when you use **gfix** on an Apollo:

- *"gfix" - name not found (OS/naming server)*
  This means that **gfix** does not show up anywhere along your shell's command search list. Check the search path, correct it if necessary, and try again.

Other error messages come from the access method. See the *OSRI Guide* for more information

See Also

Chapter 3, *Database Maintenance*.

If you write your own utility to perform some of the functions of **gfix**, see the *OSRI Guide*.

Gltj

Function

The **gltj** utility is the journal server. It can be run interactively from a window or a terminal using the **debug** option. When you run **gltj** with the **debug** option, it recognizes the options listed below. If you do not run **gltj** interactively, you must interact with it through **gcon**.

Syntax

```
gltj [options]
```

Options

q[ueue]
Queues a file or device for journaling to disk. Never journal to the same disk with the database.

c[lose]
Closes the current journal file or device.

s[tatus]
Displays the status of the journal server. Status information includes the state of journaling, the name of the output file, and known databases and connections. This information is stored in the journal database.

sh[utdown]
Shuts down the journal server. This option currently disconnects all processes running against it.

su[spend]
Suspends journaling temporarily.

r[esume]
Resumes journaling that has been suspended.

h[elp]
Displays **gltj** options with a brief description.

Troubleshooting

You may encounter the following messages when you use **gltj** on an Apollo:

- *"gltj" - name not found (OS/naming server)*
  This means that **gltj** does not show up anywhere along your shell's command search list. Check the search path, correct it if necessary, and try again.

- *operating system directive failed*
  no (library/MBX manager)
  communication error with journal "journal_directory_name"

Although a journal has been defined for the database, there is no active journla server. Use **gltj** to start the server.

See Also

See the entries in this chapter for **gcon**.

See Chapter 4, *Journaling*.

Grst

Function

The **grst** utility lets you restore a database from the after-image or long-term journal.

Syntax

```
grst [option]
```

Option

u[**ntil**] *"timestamp"*

Restores a database by rolling forward to a specified time. When specifying the timestamp, use the DD-MMM-YY format for the date, and the HH:MM:SS format for the time. Be sure to enclose the timestamp in quotes. **Grst** prompts you at each commit to apply the transaction.

If you do not specify a time, **grst** defaults to the beginning of the day you specified. This option is only available on UNIX systems.

Example

The following example restores a database just before the 2:10 disaster occurred:

O/S	Command
UNIX	% grst -u "01-mar-90 14:05:00"

Troubleshooting

You may encounter the following message when you use **grst** on an Apollo:

- "grst" - name not found (OS/naming server)

  This means that **grst** does not show up anywhere along your shell's command search list. Check the search path, correct it if necessary, and try again.

See Also

See the entries in this chapter for **gcon**.

Chapter 4, *Journaling*.

A

Accessing
  remote databases 2-6
After-image journaling
  components 4-2
  enabling with **gfix** 3-6
  overview 3-6
  see also Journaling
Apollo
  journaling 4-5
  restoring a database 2-7
  shadow 5-3, 5-11
  WBAK 2-3

B

Backup database, see **gbak**
BACKUP, VMS back up utility 2-3
Bridge, see V3.0 Release Notes

C

Checkpoint, recreating database from 4-23
Command line options, see Switches
commit
  gfix 3-8
console.addr 4-5
Corrupt database repair 3-4
Customer support information 3-5, 6-19

D

Data integrity
  overview 3-3
Database
  administration overview 1-1
  backup, see **gbak**
  changing page size 2-9
  file specifications 6-5
  maintaining 3-1, 6-18–6-22
  multi-file, see also Multi-file database
  operations overview 1-1
  page size 2-9
  recovery overview 2-1
  recreating from checkpoint 4-23
  remote 2-6
  repairing corrupt 3-4
database
  administration overview 1-1
  operations overview 1-1
date timestamp 4-12
Debug, see **gltj**
define shadow 5-4, 5-6
delete shadow 5-9
Disabling journaling 4-8, 4-19
Disaster recovery, see Shadowing, Journaling
Disk shadowing, see Shadowing

E

Errors
  gbak 6-7–6-9
  gcon 6-11–6-13
  gcsu 6-16–6-17
  gfix 6-21–6-22
  gltj 6-24–6-25
  grst 6-26
  shadow 5-12

F

Fax number for InterBase 3-5, 6-19
File specifications
  database 6-5

G

gbak
  advantages 2-3
  default device 2-5
  errors 6-7–6-9
  examples 6-5–6-7
  heterogeneous networks 2-5
  no_validity option 2-8
  options 6-2–6-5
  overview 2-1, 6-2
  tape 2-5
  troubleshooting 6-7–6-9

user defined functions 2-8
XDR (transportable) format 2-5
gcon
definition 4-2
errors 6-11–6-13
options 4-8, 6-10
overview 6-10
troubleshooting 6-11–6-13
gcsu
errors 6-16–6-17
options 6-14–6-15
overview 6-14
troubleshooting 6-16–6-17
gds_$attach_database 3-10
gds_$reconnect_transaction 3-10
gds_$transaction_info 3-10
gfix
activating 5-11
committing 3-8
errors 6-21–6-22
examples 6-20–6-21
housekeeping option 3-2
limbo transactions 3-7–3-9
options 3-2, 3-8, 6-18–6-20
overview 3-1, 6-18
sweep option 3-2
transaction recovery 3-7
troubleshooting 6-21–6-22
two-phase commit 3-7
gltj
-debug option 4-5
debugging 4-5
definition 4-2
errors 6-24–6-25
options 6-24
overview 6-24
starting on UNIX 4-5
starting on VMS 4-15
troubleshooting 6-24–6-25
grst
date timestamp 4-12
definition 4-2
errors 6-26
example 6-26
option 6-26
overview 6-26
troubleshooting 6-26

H

Heterogeneous networks
using **gbak** on 2-5
housekeeping, **gfix** option 3-2

I

interbas.jrn 4-2
InterBase
fax number 3-5, 6-19

J

journal.addr 4-5
journal.gbak 4-6
journal.gdb 4-6
journal.log 4-5
Journaling
advantages and disadvantages 4-4
components 4-2
diagram 4-3
disaster recovery 4-10
files used 4-5
monitoring status 4-16
overview 4-1
permanent journal file 4-2
restoring to time and date 4-12
UNIX 4-5–4-13
see also UNIX Installation Guide
VMS 4-14–4-25
see also **gltj**, After image journaling

L

Limbo transactions 3-7–3-9
list, **gfix** option 3-8
Long-term journaling, see After image journaling

M

Moving a database 2-5
Multi-file database
  accessing 6-5
  creating 2-10
  primary file 6-5
  secondary file 6-5

N

Network
  accessing databases on 6-5
no_validity 2-8

O

ODS, see On-disk structure
On-disk structure (ODS) 2-11
Options, see Switches
Orphan pages 3-3
Overview
  database operations 1-1
  database recovery 2-1

P

Page size
  changing 2-9
  examples 2-9
  guidelines 2-9
Performance
  changing page size to improve 2-9
  improvement 2-3
Primary file, see also Multi-file database
prompt, **gfix** option 3-8

R

rdb$files 5-4–5-5
Recovery
  automatic 3-7
  database 2-1
  gfix 3-7
  repairing corrupt database 3-4
  see also Journaling, Shadowing, Two-phase commit
  transaction 3-7
  UNIX 4-10–4-13
  VMS 4-21–4-23
Remote database
  accessing 2-6
  backing up 2-6
Repairing corrupt database 3-4
Restoring a database
  Apollo 2-7
  see also **gbak**
  UNIX 2-7
  VMS 2-7
  without **valid_if** 2-8
Restoring journal files, see **grst**

S

Secondary file, see also Multi-file database 6-5
Shadow file, see Shadowing, defining
Shadow set, see Shadowing, multi-file
Shadowing
  activating 5-11
  adding file 5-7
  advantages and disadvantages 5-3
  components 5-2
  defining 5-4
  deleting 5-9
  errors 5-12
  multi-file 5-6–5-7
  overview 5-1, 5-3
  transaction considerations 5-10
  troubleshooting 5-12
Subtransaction 3-7
Sun
  file specifications 6-5
  journaling 4-2
sweep, **gfix** option 3-2
Switches
  gbak 6-2–6-5
  gcon 6-10
  gcsu 6-14–6-15
  gfix 6-18–6-20
  gltj 4-5, 6-24

grst 6-26

T

Tape backup 2-5
tar 2-3
TCP/IP
  journal files 4-5
Transaction
  automatic recovery restrictions, see also **gfix**
  recovery 3-7
  recovery restrictions, see also **gfix**
Transportable backup (XDR format) 2-5
Troubleshooting
  gbak 6-7–6-9
  gcon 6-11–6-13
  gcsu 6-16–6-17
  gfix 6-21–6-22
  gltj 6-24–6-25
  grst 6-26
  shadow 5-12
Two-phase commit 3-7

U

UNIX
  backing up to tape 2-4
  journaling 4-5–4-13
  remote database access 2-6
  restoring a database 2-7
  shadow 5-4, 5-11
  tar 2-3
User defined function
  using **gbak** with 2-8
Utilities
  list 1-3
  see **gbak**, **gcon**, **gcsu**, **gdef**, **gfix**,
  see **gltj**, **gpre**, **grst**

V

Validating a database 3-3
VMS
  backing up 2-4
  backing up database files 2-3
  BACKUP 2-3
  disaster recovery 4-21
  journal files 4-17
  journaling 4-2, 4-14–4-25
  multi-file database 6-5
  remote database access 2-6, 6-5
  restoring a database 2-7
  shadow 5-4, 5-11