Update Guide

Borland®

InterBase® 2007

Borland Software Corporation
100 Enterprise Way, Scotts Valley, CA 95066-3249

Printed in the U.S.A.

INT0080WW21000 1E0R0403
0304050607-9 8 7 6 5 4 3 2 1

Contents

Tables

Figures

Chapter

1

InterBase 2007 Updates

This Update Guide details the various features available in InterBase 2007. InterBase 2007 introduces a new ODS version 12.0 for new InterBase databases. Please read through the chapter "Service Pack 2 Updates" to review the latest changes to InterBase 2007. Use the following links to jump to detailed information on updates available in InterBase 2007:

Chapter

2

License Changes

InterBase 2007 uses the same license manager as other Borland products, to simplify the registration and licensing processes.

This chapter describes the new process for registering your InterBase license.

Comparison

The following list describes the differences between the previous and current versions:

- You can no longer use the InterBase license file, ib_license.dat, from previous releases. These have been replaced with a serial number for registration.
- IBConsole, iblicense and other applications that depended on the InterBase License API (iblicense.dll) are obsolete for InterBase 2007 licensing. However, IBConsole is still part of the product and supports licensing for previous versions. Use the newly provided License Manager GUI tool (LicenseManager.exe) for license administration with InterBase 2007. Note that you must choose the File | Save command after you enter your serial number into the License Manager. Otherwise, the 15-day grace period will not be enabled. See the 'IBConsole' chapter of this guide for more information.
- The Sanctuary License Manager Client library is changing its name from libborland_lm.{dll,so} to sanctuarylib.dll (Windows) and libsanctuary.so (Linux, Solaris)
- When installing InterBase as a "Client Only" package, you do not need to install ib_license.dat with remote client access capability. Client-side licensing is no longer required.

InterBase Product Installation

When you install InterBase 2007, a registration wizard will come up at the end of the installation process and ask for a serial number and a license key.

Registration

After you've installed InterBase 2007, you will need to enter a Serial Number (S/N) and key provided by Borland the first time you run the server. (The client does not require a license key.)

You can register online through the Installer, or offline by running the standalone client (LicenseManager.exe).

Once you have registered your copy of InterBase 2007, you will notice a borland.lic file in your <interbase>/license directory (ILD). On startup, if valid registration details are not found, the InterBase server will report an error indicating this in the log file.

If you exit from the registration process, InterBase 2007 will function as a trial version for 15 days. If you don't complete the registration within that time, the application will stop working.

Chapter

3

Service Pack 2 Updates

Please note the following additions and updates in SP2.

Server/Client Version Compatability

2007SP2 client library fixes bugs with connecting to the older version of InterBase, it is recommended that all IB 2007 clients be upgraded with the new IB 2007 SP2 client library. For local access it is required that the local InterBase client and the InterBase server (on the same machine) be the same version. The new InterBase client library allows connections to older version of the server however only connections to the IB 2007 and IB 2007 SP2 are certified and supported. Please use the appropriate InterBase client library to connect to an older version of InterBase server, i.e. IB 7.5 client to connect to IB 7.5 server and so on. This does not apply to remote or local loop back connections which are achieved using TCP/IP.

Note The ability of IB clients to connect to older local Servers could be disallowed later releases.This does not apply to TCP/IP connectivity.

Tar Install for Linux and Solaris

Use the following files to install InterBase 2007 SP2 on Linux:

- InterBaseC_LI-V2007.tar
- InterBaseSS_LI-V2007.tar
  - setup
  - License.txt

Use the following files to install InterBase 2007 SP2 on Solaris:

- InterBaseC_SO-V2007.tar
- InterBaseSS_SO-V2007.tar
  - setup
  - License.txt

Setup Instructions

1 Execute the setup script.

2 Choose to install Server and Client or Install Client only.

3 Follow the prompts to configure:

- Installation directory
- Multi-instance option
- Instance name and port
- Run InterBase as a service

The script installs InterBase to the chosen location sets the correct variables to allow InterBase to run remotely and link to InterBase libraries.

Note The installer must be run with "root" privileges.

Note command line registration is not available.

Features and Updates

The following updates/changes have been implemented as of this release. Click the link for detailed descriptions:

- "Journal Preallocation"
- "JDBC URL Parameters"
- "Database Write Mode Default SYNC"
- "Database File Preallocations"
- "PT_BR Collation For Brazilian Portuguese"
- "Query Optimizer Improvements"
- "UDF Descriptors"

Chapter

4

Incremental Backups

The ability to create incremental backups (also called online dumps) provides you with an efficient method to backup a database between large maintenance backup sessions.

InterBase's GBAK feature (included in previous releases for full database backups and restores) fetches all the rows of the database under transaction control and writes them to backup files. Database restore reads those backup files and reconstructs a new instance of the database. This database restoration provides many useful side-effects such as rebalanced indices and packed data pages, as well as resetting the database's next transaction ID.

However, backing up large databases can take a very long time, made even longer when the database load is very heavy. During this time, GBAK's open transaction causes long record version chains to form for update-intensive rows or deleted stub rows to linger so that GBAK can read the version of the row before it was deleted.

The Incremental Backup (online dump) feature is a physical backup mechanism. It backs up the physical pages of the database to “dump” files. The incremental backup feature ensures that the output dump files represent the on-disk state of the database as of the instant the online dump was started, so transaction and page consistency are maintained in the process.

You can use the incremental backup as a staging area from which a logical GBAK can be performed, so that your production database is not adversely affected. To do this, send the online dump to a remote machine and do the logical GBAK backup on that remote machine. This will also allow you to run a database validation because validation requires exclusive database access, which cannot be obtained on a production database unless that database is shut down.

Additionally, this feature allows you to create incremental dumps that write only those database pages to the dump files that have been modified since the last time a full/incremental dump was successfully completed. It also provides a means for

you to move or copy a multi-file database to a different location. This was not possible previously because there was no way to modify the encoded file names stored on the primary and secondary header pages of the database files.

Note: This feature is only available for ODS 12 databases.

Creating Incremental Backups

Incremental Backup (online dump) support has been added to the GBAK utility using database parameter blocks (DPB).

(These DPBs are documented in this chapter so that if you are a third-party tool provider, you can add this same support for your tools.)

GBAK has two major options:

```
GBAK {-B} {options}               - backup a database to a file(s)
GBAK {-C | -R} {options}        - create or replace database from a file(s)
```

Incremental Backup (online dump) adds a third major option to GBAK:

```
GBAK {-D} {-OV} dbname   file [size] add_file1 [size1] add_file2 [size2]
...
```

The first dump file in the list is similar to the first database file in a multi-file database. It is the file that is used as a reference to an existing online dump. If there are additional dump files listed on the GBAK command line, those files are added to the set of files in the online dump.

```
[E:/tpc_c] gbak -d tpc_c.gdb tpc_c.gdmp tpc_c.gdmp.1
gbak: WARNING: Dumped 46270 pages of a total 46270 database pages
gbak: WARNING:     Dumped 1 pages to page appendix file


[E:/tpc_c] gbak -d tpc_c.gdb tpc_c.gdmp tpc_c.gdmp.1
gbak: ERROR: I/O error for file "E:\TPC_C\TPC_C.GDMP.1"
gbak: ERROR:     Error while trying to create file
gbak: ERROR:     The file exists.

gbak: Exiting before completion due to errors

[E:/tpc_c] gbak -d tpc_c.gdb tpc_c.gdmp tpc_c.gdmp.2
gbak: WARNING: Dumped 2 pages of a total 46270 database pages
gbak: WARNING:     Dumped 0 pages to page appendix file
```

In the example above, tpc_c.gdmp.1 was added in the course of a full database dump.

Re-executing the command gives an error because it tries to add tpc_c.gdmp.1 again causing a file creation error. The last command adds a new file tpc_c.gdmp.2 successfully.

The online dump files can be on either a local or a remote file system that is writable by the InterBase server. This implies that the database pages are never retrieved by the GBAK utility; online dump is a server-side operation only. While

the online dump files can be located on any mounted file system, the page appendix file is always on the local file system. This file is written to by concurrent server threads handling client requests when it is necessary to preserve the state of page's image for the online dump. This is analogous to InterBase's multigenerational architecture (MGA) where a previous version of a row is stored when updating a row to preserve a transaction's snapshot. The page appendix file helps to maintain the physical page snapshot of the online dump. It is a temporary file and is deleted when the online dump completes.

The [size] parameter is optional and denotes the file's size in units of pages, using the database's page size. If the [size] parameter is not provided then that dump file's size will be determined by its file-sequenced counterpart in the database. If the dump file's sequence is higher than the sequence of any database file then it takes the size of its predecessor dump file.

If you run GBAK -D against an existing online dump, an incremental dump will be created.

```
[E:/tpc_c] gbak -d tpc_c.gdb tpc_c.gdmp
gbak: WARNING: Dumped 46270 pages of a total 46270 database pages
gbak: WARNING:     Dumped 23 pages to page appendix file

[E:/tpc_c] gbak -d tpc_c.gdb tpc_c.gdmp
gbak: WARNING: Dumped 2 pages of a total 46270 database pages

     gbak: WARNING:     Dumped 0 pages to page appendix file
```

This updates the online dump with only those pages that have changed since the last dump. An incremental dump can always be retried if it fails. If a full online dump fails, InterBase will delete the online dump files that were written prior to the failure. If InterBase cannot not access those files because of the failure, those online dump files will have to be deleted manually.

Over-writing Incremental Backups

The -OV overwrite switch causes the current set of online dump files to be deleted, and initiates a full database dump.

```
[E:/tpc_c] gbak -d tpc_c.gdb tpc_c.gdmp
gbak: WARNING: Dumped 2 pages of a total 46270 database pages
     gbak: WARNING:     Dumped 1 pages to page appendix file

[E:/tpc_c] gbak -d -ov tpc_c.gdb tpc_c.gdmp
gbak: WARNING: Dumped 46270 pages of a total 46270 database pages
gbak: WARNING:     Dumped 7 pages to page appendix file
```

The online dump files are marked as a read-only InterBase database. This means that it can be accessed by read-only database applications. It is undefined how such database applications will behave if they access the online dump “database” while the dump files are being incrementally updated. If an online dump is converted to read-write, it ceases to be an online dump and becomes a standalone database. Attempting to perform an online dump against it will fail.

```
[E:/tpc_c] gfix tpc_c.gdmp -mode read_write
```

```
[E:/tpc_c] gbak -d tpc_c.gdb tpc_c.gdmp
gbak: ERROR: online dump failure: dump file has no dump timestamp
gbak: Exiting before completion due to errors

[E:/tpc_c] gfix tpc_c.gdmp -mode read_only

[E:/tpc_c] gbak -d tpc_c.gdb tpc_c.gdmp
gbak: ERROR: online dump failure: dump file has no dump timestamp
gbak: Exiting before completion due to errors
```

There is no online dump restore operation, per se. The online dump can be converted to a read-write database, as mentioned above, and used in place. If the current location is not convenient for database processing then online dump can be run against these dump files to copy them somewhere else local or remote. This provides a general copy mechanism that allows multifile databases to be copied and have their internal secondary file name links automatically updated for the copy destination.

Database validation (GFIX -V) can be run against an online dump because it is a database. An additional validation check is performed against an online dump, which checks that no database page has a write timestamp greater than that of the online dump timestamp. The online dump timestamp represents that last time a full or incremental dump succeeded.

```
[E:/tpc_c] gfix -v -n tpc_c.gdmp
Summary of validation errors

       Number of database page errors  : 1
         and in the InterBase log file:.
IBSMP (Server)    Sat Jun 24 14:41:36 2006
   Database: E:\TPC_C\TPC_C.GDMP
   Page 155 has timestamp 1151170444 greater than dump timestamp 1151170438
```

Timestamp Changes

GSTAT -H has been modified to list the online dump timestamp after the database creation date entry. Note that the database creation date is that of the source database and not the online dump.

```
[E:/tpc_c] gstat -h tpc_c.gdmp

Database "tpc_c.gdmp"

Database header page information:

        Flags                   0
        Checksum                12345
        Write timestamp         Jun 28, 2006 19:57:41
        Page size               4096
        ODS version             12.0
        Oldest transaction      72
        Oldest active           73
        Oldest snapshot         73
        Next transaction        74
        Sequence number         0
        Next attachment ID      0
        Implementation ID       16
        Shadow count            0
        Page buffers            0
```

```
Next header page        0
Clumplet End            102
Database dialect        3
Creation date           Jun 25, 2006 13:22:10
Online dump timestamp   Jun 28, 2006 19:59:16
Attributes              read only
```

Variable header data:

```
Dump file length:               20000
*END*
```

You can request an online dump by passing a string of database parameter blocks to the isc_attach_database() API.

The following table lists the names and values of database parameter blocks (DPB) used to access this feature. All general requirements and restrictions for DPB construction as documented in the InterBase API Guide apply here.

Table 4.1 Database Parameter Blocks (DPBs)

Parameter Name	Purpose	Length	Value
isc_dpb_online_dump	Directive to initiate an online dump	1	0 or 1
isc_dpb_old_overwrite	Indicates the current online dump files should be deleted and a full database dump executed (optional)	1	0 or 1
isc_dpb_old_file_name	String specifying the name of an online dump file, up to 255 characters	No. of bytes in string	Dump file name string
isc_dpb_old_file_size	Number of pages for online dump file (optional)	No. of bytes for length indicator (1, 2, or 4)	No. of pages for dump length

A successful online dump returns a warning status vector to pass back dump information status:

```
status [0]  = isc_arg_gds
status [1]  = isc_arg_success
status [2]  = isc_arg_warning
status [3]  = isc_old_dump_stats
status [4]  = isc_arg_number
status [5]  = <no. of dumped pages>
status [6]  = isc_arg_number
status [7]  = <total no. of DB pages>
```

```
status [8]  = isc_arg_gds
status [9]  = isc_old_appendix_stats
status [10] = isc_arg_number
status [11] = <no. pages written to appendix>
status [12] = isc_arg_end
```

Page Appendix File

When an online dump is running, client worker threads never write to the online dump files. Thus, their performance is not degraded by writing over the network to a remote file system. However, to maintain physical and time consistency of the dump, client worker threads may write pages to a local temporary file with a prefix of “ib_dump_”. Any database page is guaranteed to be written at most one time to this temporary file. This temporary file is known as the dump or page appendix file.

For very large databases with intensive update activity, the page appendix file could also grow to a very large size. There must be adequate space in the temp directories to handle this storage demand or the online dump will fail. The dump information returned to GBAK about the number of pages written to the appendix file can aid configuration of the temp file space.

Incremental Backup Guidelines

- Since an online dump is a physical backup technique, the online dump files are not transportable to other hardware platforms. It would be necessary to use GBAK's traditional logical backup on the online dump to transport it.
- Multiple online dumps of the same or distinct databases can be run concurrently though this would not be recommended for performance reasons.
- Performing an incremental online dump still requires a full scan of the source database.
- The performance improvement accrues from limiting the number of page writes to the online dump files, especially if those files are located on a remote file server.
- This feature is used internally by InterBase to create an online dump of the database to a journal archive directory when CREATE JOURNAL ARCHIVE is executed.
- An active online dump can be cancelled by the InterBase Performance Monitor or killing the GBAK process.
- External tables are not backed up by an online dump.
- External tables may not be accessible if the online dump is attached as a read-only database. If the external file pathnames can't be accessed from the online dump's location, there is no way to modify the dump's metadata without making the dump a read-write database. If it is made a read-write database, it can no longer be a target for online dump again.

Chapter

5

Journaling

This chapter describes the journal subsystem and the DDL syntax used to create, alter, and drop journal files and journal archives. Database journaling improves VLDB management and facilitates disaster recovery.

Note that Journaling is only available on the Server Edition of InterBase 2007, and not on the Desktop Edition.

The Journal Subsystem

The following criteria should be used to determine the optimal journaling configuration:

- The I/O speed of the device on which the journal files are created.
- The speed of concurrent creation of new journal files.
- Hardware requirements and ease of setup.

To improve performance, it is recommended that database files and journal files be created on different devices. The default behavior of CREATE JOURNAL creates the journal files in the same location as the database file. While this is not a recommended practice, it can be advantageous when the database files can be cached in main memory. In this case there would be no database read operations and only minimal database writes during journal checkpoints, which can be configured to occur infrequently.

The CREATE JOURNAL statement causes all subsequent write operations on a database to be done asynchronously. The journal file I/O is always synchronous and cannot be altered. All transaction changes are safely recorded on durable storage before the transaction is committed. This guarantees the ACID properties of a transaction (the database industry standards for Atomicity, Consistency, Isolation, and Durability).

Using asynchronous I/O for database writes allows the operating system to optimize file I/O, such as by writing consecutive pages together, or by using scatter/gather techniques that write consecutive pages in discontiguous page buffers. Journal file I/O is performed using InterBase's careful write strategy. This implies that database pages can be written back to the database in any order after their changes have been journaled.

During a database checkpoint, any database page writes that were buffered asynchronously are flushed to disc before checkpoint completion is signaled. You can re-enable synchronous writes for the database, which will remove the requirement for a flush operation before a database checkpoint can be considered done.

Journal Archives

A journal archive is the set of destination directories that will hold the current set of journal files for a particular database. For disaster recovery purposes, a journal archive should always be located on a server machine or file server remote from the database server. At the current time, it is a requirement that all journal files must be in the same directory.

There are four types of journal archives: Database Archive, Journal Archive, Recovery (also known as Online Dump, or Point-in-Time Recovery) and Archive Sweep.

It is not necessary for InterBase to be installed and running on the machine used for journal archive storage. Since the journal archive appears as a remote file server, dissimilar platforms can serve as a remote journal archive. For example, a Linux database server could NFS mount a file system on a Solaris file server or NetWare.

Enabling Journal Files

This section shows the DDL statements required to enable journaling for a database.

Creating Journal Files

Creating a journal requires exclusive access to the database. The DDL syntax is:

CREATE JOURNAL [<journal-file-specification>] [LENGTH <number-of-pages> [PAGES]]

```
[CHECKPOINT LENGTH <number-of-pages> [PAGES]]
[CHECKPOINT INTERVAL <number-of-seconds> [SECONDS]]
[PAGE SIZE <number-of-bytes> [BYTES]]
[PAGE CACHE <number-of-buffers> [BUFFERS]]
[[NO] TIMESTAMP NAME];
```

The <journal-file-specification> is a quoted string containing the full path and base file name of the journal file. The base journal file name is used as a template for the actual journal file names as they are created. The form of the actual journal file name is discussed in detail below.

The LENGTH clause specifies the number of pages that will be written to the journal file before initiating a rollover to a new journal file. A single journal file is limited to 2GB in size.

Several options control the journaling configuration of a database. These options are described in Table 5.1, “CREATE JOURNAL Options”.

Table 5.1 CREATE JOURNAL Options

Option	Description
CHECKPOINT LENGTH	Determines the number of journal pages to be written before initiating a database checkpoint.
CHECKPOINT INTERVAL	Determines the number of seconds between database checkpoints. Note: If both CHECKPOINT LENGTH and CHECKPOINT INTERVAL are specified, whichever event occurs first will initiate a database checkpoint.
PAGE SIZE	Determines the size of a journal page in bytes. A journal page size must be at least twice the size of a database page size. If a journal page size of less is specified, it will be rounded up to twice the database page size and a warning will be returned. The journal page size need not be a power of 2.
PAGE CACHE	Determines the number of journal buffers that will be allocated. The size of each buffer is the same as the journal page size.
[NO] TIMESTAMP NAME	Determines whether or not to append the file creation timestamp to the base journal file name. If this option is on, the base journal file name will be appended with a timestamp of the form: <YYYY>_<MM>_DD>T<hh>_<mm>_ss>Z.<sequence-number>.journal

All CREATE JOURNAL clauses are optional. The default values are shown in Table 5.2, “CREATE JOURNAL Options - Default Values”.

Table 5.2 CREATE JOURNAL Options - Default Values

Option	Default Value
<journal-file-spec>	The full database path and file name
LENGTH	4000 pages
CHECKPOINT LENGTH	3500 pages
CHECKPOINT INTERVAL	0 seconds
PAGE SIZE	Twice the database page size
PAGE CACHE	100 buffers
TIMESTAMP NAME	Enabled

Disabling Journal Files

The DROP JOURNAL statement will discontinue the use of write ahead logging and delete all journal files. This operation will not delete any journal files in the journal archive but will discontinue maintenance of the journal archive. Dropping journal files requires exclusive access to the database. The syntax of this statement is:

```
DROP JOURNAL
```

Journal Archives

The purpose of the Journal archive is to support long-term database recovery. This feature provides for disaster recovery in the event a database becomes unavailable due to hardware or software failures that may make the primary database permanently inaccessible.

The journal archive does not automatically copy journal files or perform online database dumps. There are no DDL clauses to declaratively specify when to backup journals to the journal archive. It is similar to logical database backup, GBAK, in that a separate utility must be run to effect the archiving of an archive.

A journal archive creation statement defines a target journal archive directory to the database. Creating a journal archive does not require exclusive database access. This is important because the side-effect of this statement is to create an online dump of the database into the journal archive.

The online dump (OLD) is a physical copy of the database that is transaction-consistent as of the start of the dump. The online dump copies the database without holding a transaction open, which will prevent database performance from suffering due to the buildup of record back versions.

Creating a Journal Archive

The DDL syntax for creating a journal archive is:

CREATE JOURNAL ARCHIVE [<journal archive directory>]

where <journal archive directory> ::= <directory specification>.

Note that the CREATE JOURNAL ARCHIVE DDL statement does not create the file system directories. The statement will return an error if the directory does not exist or is not accessible.

The journal archive directory-specification should be specified such that it is accessible for a file copy operation. For example, if the archive directory is a UNIX symbolic link, use the symbolic link and not the target path name. The directory can be specified as a UNC path, as long as the underlying file APIs can open the file using that specification.

If a journal archive directory specification is not given, the journal directories themselves become a de-facto archive. Normally, when a database checkpoint that writes to the database what has been recorded in the journal files, the current journal files of the database are deleted. The following DDL statement:

CREATE JOURNAL ARCHIVE;

will mark all database journal files so that they will not be deleted when a checkpoint occurs. This also means that no copying is required, since the files are already where they belong for archiving purposes.

The number-of-pages parameter specifies the number of pages to be written before an archive directory spill-over occurs. When a directory spill-over occurs, the next archive directory will be used for copy operations.

Dropping a Journal Archive

The DROP JOURNAL ARCHIVE statement disables journal archiving for the database. It causes all journal files and database file dumps to be deleted in all journal archive directories. The file system directories themselves are not deleted.

Disabling journal archiving does not disable database journaling. The database will continue to use the write-ahead protocol to commit database changes to the journals. If the intent is to also disable short term journaling, then a separate DROP JOURNAL statement must be executed. The DDL syntax is:

DROP JOURNAL ARCHIVE

Managing Journal Archives

Archived database dumps representing the starting point from which long-term database recovery is initiated. A set of archive journal files will be applied to a copy of the archive database in the same way that local journal files are applied to a production database during short-term recovery. Optionally, an InterBase timestamp can be specified (-until <timestamp>) to indicate a point-in-time until which the journal files will be applied.

Recovery

When the archive is used to recover a database, the resulting database is not a journaled database. This means that RDBLOG_FILES, RDBJOURNAL_FILES and the log page of the database are empty. This is to prevent the database from accidently using the journal and journal archive of an existing database.

Database recovery is usually used when the original database is corrupted or unavailable due to hardware failures. However, it is possible to recover a database on the same machine as the working, production database or on a different machine where the journal and journal archive directories have no similarly-named directories. Therefore, if you want to use journaling and/or journal archiving for a recovered database, it is necessary to execute the appropriate DDL commands to do so.

Archiving and Recovery Commands

Use the gbak command to archive databases and journal files to the archive, and also to recover a database from the archive and load it back to a specified local directory.

To archive a database:

```
gbak -archive_database <dbname>
```

To archive local journal files:

```
gbak -archive_journals <dbname>
```

To recover a database (optionally to a point-in-time):

```
gbak -archive_recover [-until <timestamp>] <archive_dbname> <local_dbname>
```

If you do not use the -until command line switch, the database recover program will apply as many journal files as possible to recover a database to the most recent point-in-time. You should put quotes around the words, 'UNTIL timestamp' if you invoke the gbak command from a shell, so the date and time components are not passed in as

separate arguments. Please refer to Embedded SQL Guide, Chapter 7: Working with Dates and Times, Section: Formatting dates for input, for a description of how to specify these timestamps. Note the use of special literals like “now” and “today”.

If possible, the database recovery program will attempt to “jump” from the archive to the local journal directory to apply the journal files that were never copied to the archive. In this way, a database may be recovered to the most recently committed transaction of the original database.

Managing Archive Size

If you allow it, the archive will grow in storage size infinitely as the database and the most current journal files are continually archived.

Use the gfix command to manage and garbage-collect archive items that are no longer required As the number of journal files grows in the archive when you have not created more recent archived database dumps, the time that you will need to recover a database from the archive also grows. Therefore, it is a good practice to periodically create additional database dumps in the archive. At some point, you may decided that older database dumps, and the journal files on which they depend, are no longer necessary, since the basis of recovery will be on more recent database dumps and journal files.

Archive Sequence Numbers and Archive Sweeping

All archived items are denoted by an archive sequence number that corresponds to the order in which the items were created in the archive.

To garbage-collect archive items less than an archive sequence number, use the archive sweep option combined with the archive sequence number:

```
gfix -archive_sweep [-force]  <archive_sequence_no>
```

If an archived item cannot be swept (garbage-collected) for some reason, the sweep will stop and return an error status. In some cases, this could stop the command from ever succeeding. For example, if an archive is manually deleted with a shell OS command, the sweep will always fail because it can't find the file to drop. The -force option continues to delete as much as possible, regardless of errors.

The -force switch will log errors to the InterBase error log instead of returning an error status.

To specific how many database dumps to allow in the archive:

```
gfix -archive_dumps <number>
```

Once the number of database dumps in the archive exceeds the <number> given, all lower sequenced archive items are deleted from the archive.

Sometimes all lower sequenced items cannot be deleted. For example, a database dump may depend on a lower sequenced journal file with which to start recovery. In that case, InterBase will automatically adjust the given sequence number to a lower number, so that this dependency is not lost.

Tracking Archive State

To track that state of the archive, a new system table, RDB$JOURNAL_ARCHIVES, has been added for ODS 12 (InterBase 2007) databases. The gbak and gfix commands listed above use this system table to decide which archive items are targets for the commands.

Table 5.3 RDB$JOURNAL_ARCHIVES Table

Column Name	Data Type	Length	Description
RDB$ARCHIVE_NAME	VARCHAR	1024	The name of the archived item.
RDB$ARCHIVE_TYPE	CHAR	1	The type of the archived item. 'D' indicates a database dump. 'S' indicates a secondary database file of a database dump. 'J' indicates a journal file.
RDB$ARCHIVE_LENGTH	INT64	8	Length of the archived item as stored in bytes.
RDB$ARCHIVE_SEQUENCE	INTEGER	4	Sequence number of archived item.
RDB$ARCHIVE_TIMESTAMP	TIMESTAMP	8	Timestamp when item was stored in the archive.
RDB$DEPENDED_ON_SEQUENCE	INTEGER	4	Sequence of archived item that this item depends on. For 'S' archive types, it would be the sequence no. of the 'D' primary database dump file. For 'D' archive types, it is the sequence no. of the starting journal file for recovering from the archive.
RDB$DEPENDED_ON_TIMESTAMP	TIMESTAMP	8	As above, but the archive timestamp for the archived item that this item depends on.

Restrictions on Journals and Archives

1. The archive is platform-specific. An archive created with InterBase for Windows cannot be directly used to recover on InterBase for Unix. Instead, an archived database dump could be logically backed up in transportable format and then logically restored on the other platform.
2. The journal and journal archive are restricted to a single directory. The number of items allowed to be archived will be limited by the number of files that are allowed in a directory for a given file system.
3. Only full database dumps are archived. In particular, it is not possible to archive incremental database dumps.
4. Journaling must be enabled for a database before the database can be configured for journal archiving.

Journal Preallocation

Use Journal Preallocation to statically preallocate space to assure the journal subsystem will not fail at runtime due to lack of disk space. With Preallocation determine journal file space requirements while simultaneously guaranteeing the space is allocated in advance.

Note If several databases are using the same disk for journalling, it may not be obvious how much total disk space is required by all databases.

Because journal files are written with synchronous I/O, each data write will cause the file system metadata to be updated to insure data consistency. This action seeks the disk heads away from the tail of the journal file and requires repositioning of the disk heads on a subsequent write. A PREALLOCATE clause is added to the syntax of the CREATE JOURNAL statement to facilitate this behavior:

Example

```
... [[NO] PREALLOCATE int [PAGES]]
```

The allocation unit is measured by journal file pages. Where each journal page has a size equal to what was specified in the CREATE JOURNAL statement or the default of twice the database page size. The default behavior if this clause is omitted is to preallocate the journal files, according to the remaining journal file specifications. In all cases, what is preallocated is what would have eventually been allocated during database operation in the absence of explicitly requesting journal file preallocation.

Journaling Tips and Best Practices

The following example is included in this guide to help you set up your own 'best configuration' for journaling. This example was designed for a minimal configuration, which will minimize journal file rollover and reduce the probability of journal buffer wait states. The default property values for the journal subsystem are for a minimal configuration, designed not to overwhelm low-end machines. This is very similar to InterBase's default page buffer cache of 2048.

We will start this example by setting the following parameters:

```
CREATE JOURNAL 'e:\database\test'
LENGTH 65000
CHECKPOINT LENGTH 10000
PAGE CACHE 2500;
```

Given a database that has an 8KB page size, the journal PAGE SIZE will default to 16KB (2 x 8KB).

Therefore, the LENGTH parameter (65000) will cause rollover to a new journal file every 1GB (65000 x 16KB). The built-in LENGTH default (500) means that your system will roll over to a new journal file every 8MB, which will be extremely frequent, and you may notice a performance drop during this process. Using a larger LENGTH value will make this occur (65000/500 or 130 times) less often.

The CHECKPOINT LENGTH parameter of 10000 means the database checkpoint will occur every 160MB (10000 x 16KB). Assume the built-in CHECKPOINT LENGTH is 500, which means your system will checkpoint the database every 8MB (500 x 16KB). CHECKPOINT LENGTH is a matter of individual taste. It represents the maximum no. of bytes that will have to be applied to a database from the journal files after a system crash. You can expect to average between 1MB to 2MB/sec. applying the journal files during the recovery process. So the 160MB checkpoint length suggested here would take a maximum of about 2 minutes to recover depending on your machine. If your organization can tolerate a longer recovery time in return for minimizing the online frequency of database checkpoints, then raise the CHECKPOINT LENGTH accordingly.

The PAGE CACHE parameter can be raised to reduce the probability of incurring journal buffer wait states. At any moment, the journal cache writer thread will be syncing some number of journal buffers to the journal file on disk. During this period, we want to insure that the worker threads have enough spare journal buffers to write to when a database page's journal changes need to be moved to a journal buffer.

For example, imagine that the journal cache writer is syncing 500 journal buffers to disk. The 2500 journal buffer configuration will leave 2000

spare buffers for the worker threads to dump their journal changes. At the built-in PAGE CACHE default of 100, your worker threads can stall due to a high rate of journal buffer wait states.

Lastly, the use of a SAN mirrored cache will always make InterBase's journaling sub-system result in lower performance than a non-journaled InterBase database. This is because twice the amount of data is being written with the journaling subsystem: once to the journal files and once to the database files, plus the additional CPU cost of journal cache management in the InterBase server.

Even for direct-attached storage, it is necessary to pay attention to on-disk write cache enablement. New computers sometimes arrive with on-disk write cache enabled. This means that sync writes to a database or journal are not really synchronized to disk oxide. Unless the write cache (SAN or direct) has been disabled or has battery backup, it can't offer durability for database commits.

InterBase journaling should only result in a performance gain when disk I/O is write-through, where every database write goes to disk oxide and not an on-disk cache.

Hopefully, the CREATE JOURNAL statement above will minimize this cost. Remember that the end goal is to provide point-in-time disaster recovery using the CREATE JOURNAL ARCHIVE statement to archive time-consistent database dumps and journal files.

You may want to have the JOURNAL and JOURNAL ARCHIVE colocated in the same directory. So you can now issue:

CREATE JOURNAL ARCHIVE <same directory as specified in CREATE JOURNAL>, or simply CREATE JOURNAL ARCHIVE.

Then,

GBAK -A(RCHIVE_DATABASE) <my_database> to create a time-consistent database dump in the journal archive.

This also causes archival of existing non-archived journals in its wake. In this case, these journal files are just marked as archived since a copy operation isn't needed when the JOURNAL and JOURNAL ARCHIVE are colocated. A row for each archived item is entered into RDB$JOURNAL_ARCHIVES.

With this method, you don’t have to worry that the copying can be halted by a checkpoint or a gbak -a command. Copying the database to its archive uses the online dump feature new to InterBase 2007. So, in the archive directory listing below, the database dump, TPC_C.2006-08-21T15-48-17Z.1.DATABASE, has no database changes made after 2006-08-21 15:48:17. It doesn't care what updates are going to the main database while it is being dumped or after it is finished

dumping. This includes the checkpoint process.

```
24 Aug 21 15:45 IB_JOURNAL
24 Aug 21 15:45 IB_JOURNAL_ARCHIVE
130399832 Aug 21 16:00 TPC_C.2006-08-21T15-45-11Z.1.JOURNAL
979562496 Aug 21 16:00 TPC_C.2006-08-21T15-48-17Z.1.DATABASE
130397262 Aug 21 16:00 TPC_C.2006-08-21T15-51-51Z.2.JOURNAL
130399932 Aug 22 18:13 TPC_C.2006-08-21T15-57-03Z.3.JOURNAL
130398336 Aug 22 18:13 TPC_C.2006-08-22T18-06-19Z.4.JOURNAL
130397418 Aug 22 18:14 TPC_C.2006-08-22T18-10-52Z.5.JOURNAL
35392721 Aug 23 00:27 TPC_C.2006-08-22T18-14-47Z.6.JOURNAL
```

A GSTAT -L TPC_C.2006-08-21T15-48-17Z.1.DATABASE shows the following:

```
Database log page information:
Creation date Aug 21, 2006 15:45:11
Log flags: 1
Recovery required

Next log page: 0
Clumplet End 907

Variable log data:
Control Point 1:
File name: E:\TPC_C_JOURNALS_AND_ARCHIVES\
TPC_C.2006-08-21T15-45-11Z.1.JOURNAL
Partition offset: 0 Seqno: 1 Offset: 5694
```

This is what the main database's log page looked like at precisely 2006-08-21 15:48:17. If you attempt to recover using this database dump it will start with journal file, TPC_C.2006-08-21T15-45-11Z.1.JOURNAL, at offset 5694 and continue through the last journal file or whatever timestamp was specified with an optional -UNTIL clause:

```
GBAK -ARCHIVE_R E:\TPC_C_JOURNALS_AND_ARCHIVES\
TPC_C.2006-08-21T15-48-17Z.1.DATABASE E:\TPC_C_RECOVER\TPC_C.GDB
-UNTIL "2006-08-21 18:08:15"
```

and in the INTERBASE.LOG:

```
IBSMP (Server) Tue Aug 22 22:49:08 2006
Database: E:\TPC_C_RECOVER\TPC_C.GDB
Long term recovery until "2006-08-21 18:08:15" begin

IBSMP (Server) Tue Aug 22 22:49:09 2006
Database: E:\TPC_C_RECOVER\TPC_C.GDB
Applying journal file:
```

```
E:\TPC_C_JOURNALS_AND_ARCHIVES\TPC_C.2006-08-21T15-45-11Z.1.JOURNA
L

IBSMP (Server) Tue Aug 22 22:51:38 2006
Database: E:\TPC_C_RECOVER\TPC_C.GDB
Applying journal file:
E:\TPC_C_JOURNALS_AND_ARCHIVES\TPC_C.2006-08-21T15-51-51Z.2.JOURNA
L

IBSMP (Server) Tue Aug 22 22:53:24 2006
Database: E:\TPC_C_RECOVER\TPC_C.GDB
Applying journal file:
E:\TPC_C_JOURNALS_AND_ARCHIVES\TPC_C.2006-08-21T15-57-03Z.3.JOURNA
L

IBSMP (Server) Tue Aug 22 22:55:44 2006
Database: E:\TPC_C_RECOVER\TPC_C.GDB
Applying journal file:
E:\TPC_C_JOURNALS_AND_ARCHIVES\TPC_C.2006-08-22T18-06-19Z.4.JOURNA
L

IBSMP (Server) Tue Aug 22 22:55:57 2006
Database: E:\TPC_C_RECOVER\TPC_C.GDB
Long term recovery end
```

GBAK -A (creating archive db dump) never locks anything. The only archive management restriction is that archive operations are serialized. You can't do multiple GBAK/GFIX operations against it at the same time. The important point here is that the main database is fully accessible at all times.

GBAK -ARCHIVE_J(OURNALS) <my_database> causes non-archived journal files to be copied to the archive (or marked as archived as above) when you don't want to dump the whole database. Again, a row is entered into RDB$JOURNAL_ARCHIVES for each archived journal file.

GBAK -ARCHIVE_S(WEEP) <sequence no.> <my_database> deletes all files in RDB$JOURNAL_ARCHIVES with RDB$ARCHIVE_SEQUENCE less than the requested sequence.

GBAK -ARCHIVE_DU(MPS) <number> <my_database> configures the maximum number of database dumps allowed in the archive. After issuing GBAK -ARCHIVE_DATABASE, archive management will automatically delete the oldest archive database dump and all earlier journal files if the dump limit has been exceeded by the addition of the new database dump.

GBAK -ARCHIVE_R(ECOVER) <archive_directory/archive_database> <new_database> [-UNTIL <timestamp>] [-BUFFERS <number>], will recover a

database from the archived journal files. Remember that <archive_directory> has to be mounted for read access on the machine performing the recovery. Archive directories can be located on InterBase servers or passive file servers and appliances. The archived files are opened directly by clients and not through an InterBase server. Archive database dumps are sealed so you can simultaneously run database validation (usually requires exclusive), logical GBAK, and have multiple, same-platform machines on the network attach the database for read-only queries, which implies high levels of page I/O over the network.

If the most current, non-archived journal files are accessible from the machine where the recover is being executed, then the recovery process will "jump" to those journal files to recover the most recently committed transactions, notwithstanding the optional -UNTIL clause. The recovered database is divorced of any journal or journal archive so it is necessary to define them again if desired.

However, it is much more useful to leave the recovered database in a perpetual state of long term recovery. That is, every time after the first GBAK -ARCHIVE_RECOVER, subsequent GBAK -ARCHIVE_RECOVER apply the incremental journal changes. This provides perfect symmetry with the online dump feature:

```
GBAK -DUMP <main_database> <dump_database> -- Full online dump
GBAK -DUMP <main_database> <dump_database> -- Incremental dump
GBAK -DUMP <main_database> <dump_database> -- Incremental dump
...
GFIX -MODE READ_WRITE <dump_database> -- Divorce from main DB

GBAK -ARCHIVE_R <main_database> <recv_database> -- Full recover dump
GBAK -ARCHIVE_R <main_database> <recv_database> -- Incremental recover
GBAK -ARCHIVE_R <main_database> <recv_database> -- Incremental recover
...
GFIX -MODE READ_WRITE <recv_database> -- Divorce from main DB

This functional modification is much more efficient. Full archive recovery
can take hours depending on the volume of journal changes.
```

If you divorce from the database, you save 1 second in not having to type GFIX -MODE READ_WRITE at the cost of having to create another full recovery if you want a more recent copy (hour(s)). Now you have to run GFIX -MODE READ_WRITE to divorce, but you gain hours of efficiency by being able to get the incremental journal changes since the last GBAK -ARCHIVE_RECOVER.

This also means that the recovered database can be deployed more quickly if the main database is lost. It also can function as a more up-to-date READ_ONLY database for queries and reporting purposes.

Lastly, the journal archive is never implicitly dropped as a side-effect of DROP DATABASE or DROP JOURNAL. It is necessary to explicitly issue a DROP JOURNAL ARCHIVE statement before DROP DATABASE. The journal archive potentially represents the last known source of the dropped database's contents so it is intentionally difficult to delete.

Chapter

6

Batch Updates

Batch updates allow you to send a group of SQL statements to a server in a single unit. Grouping SQL statements into batches reduces the amount of network traffic between the client and the database server. This results in improved performance, especially in LAN and WAN environments.

This chapter describes how to use batch updates with InterBase 2007.

Note Batch updates only work using the InterBase 2007 client library and InterClient JDBC driver.

Using Batch Updates

You can send multiple INSERT, UPDATE, and DELETE statements to the server using batch updates. In response, the server returns an array of ULONG values that reflect the number of affected rows per statement.

SQL query statements like SELECT are not supported in batch updates. SQL DDL is supported, however, the CREATE DATABASE statement is not.

Batch updates decrease the amount of communication between client and server, thereby improving performance in a LAN or WAN environment. Figure 6.1 shows the flow of communication between client and server when completing a number of INSERT statements using traditional InterBase client APIs. Note the flow of communication shown in the figure also applies to UPDATE and DELETE statements.

Figure 6.1 INSERT Without Batch Updates

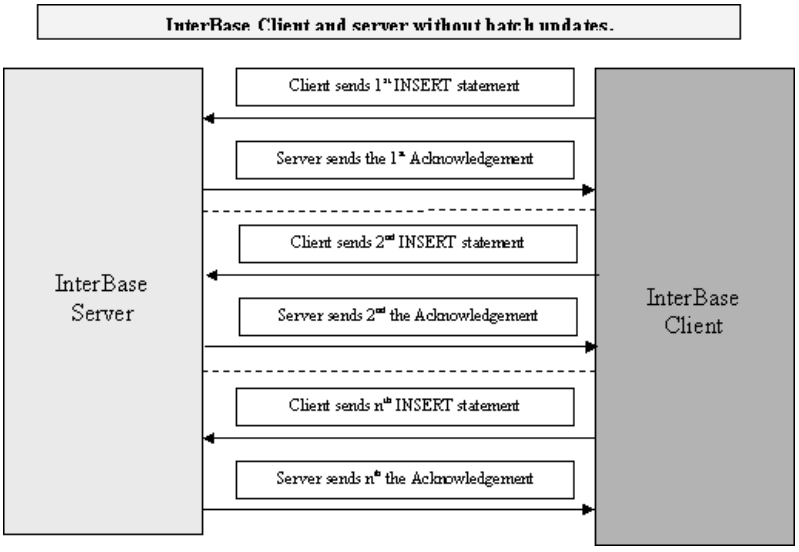
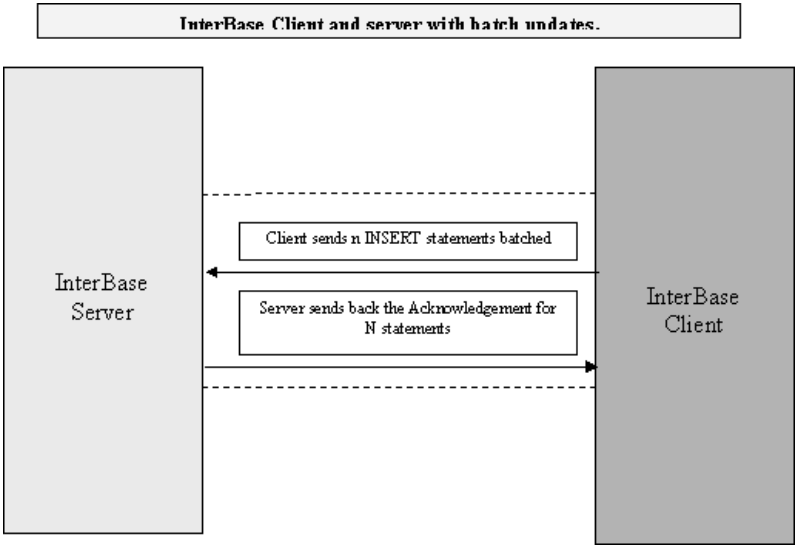


Figure 6.2 shows the flow of communication when using batch updates. Note the reduction in network traffic, resulting in better performance.

Figure 6.2 INSERT With Batch Updates



Client APIs for Batch Updates

The InterBase client library now contains two additional functions to support batch updates: One function is used for immediate batch execution. The second function is used for execution of prepared SQL statements containing parameters.

With both prepared and non-prepared batch updates, the client application must explicitly commit or rollback changes caused by the batched SQL statements. Batch updates will not work if auto commit mode is on.

The isc_dsql_batch_execute_immed Function

The new isc_dsql_batch_execute_immed function is used to execute a group of INSERT, UPDATE, DELETE or DDL commands. Its signature is:

```
ISC_STATUS isc_dsql_batch_execute_immed(ISC_STATUS *status_vector,
isc_db_handle *db_handle, isc_tr_handle *tr_handle, int dialect,
ULONG number_of_sql, char[] *sql, ULONG *rows_affected);
```

The meaning of each parameter is explained in Table 6.1.

Table 6.1 isc_dsql_batch_execute_immed Parameters

Argument	Description
status_vector	The address of an array of type ICS_STATUS. The respective ISC_STATUS values for each SQL statement in the batch will be returned in this array.
db_handle	The address of the database handle.
tr_handle	The address of the transaction handle.
dialect	The SQL dialect to use for the statements in the batch update.
number_of_sql	The number of SQL statements included in the batch update. This argument reflects the number of strings in the sql array.
sql	An array of NULL-terminated strings. Each string is an SQL statement to execute in the batch update. The SQL statements do not need to be terminated with semicolons. Instead, each SQL statement is terminated by a C NULL character.
rows_affected	A pre-allocated array of type ULONG, which will be used to store the number of rows affected by the corresponding SQL statement in the sql array.

If a statement fails, the ISC_STATUS will be set, and the corresponding entry in the rows_affected array will be set to -1, or 0xFFFFFFFF.

New Error Conditions

Table 6.2 shows the error codes are returned by the isc_dsql_batch_execute_immed function.

Table 6.2 Batch Update Error Codes

Error code	Description
isc_string_too_large	Returned when the total length of all SQL statement strings (including NULL characters) exceeds 65325.
isc_dsql_select_in_batch	Returned when one of the SQL statements is found to be a SELECT statement. Note that all statements prior to the SELECT will be executed. The rows_affected argument for the SELECT statement will be set to -1. The changes made by statements prior to the SELECT are not committed or rolled back. This needs to be done specifically by the application or driver.

The isc_dsql_batch_execute Function

The new isc_dsql_batch_execute function supports batch updates for prepared SQL statements. This function allows you to batch the data used in a parameterized SQL statement. The signature of the function is:

```
ISC_STATUS isc_dsql_batch_execute(ISC_STATUS *status_vector,
isc_db_handle *db_handle, isc_tr_handle *tr_handle,
isc_stmt_handle stmt_handle, int dialect, USHORT number_of_rows,
XSQLDA *insqlda, XSQLVAR[] *batch_vars, ULONT *rows_affected);
```

Table 6.3 shows the meaning of each argument.

Table 6.3 isc_dsql_batch_execute Parameters

Argument	Description
status_vector	The address of an array of type ICS_STATUS. The respective ISC_STATUS values for each SQL statement will be returned in this array.
db_handle	The address of the database handle.
tr_handle	The address of the transaction handle.
stmt_handle	Statement handle previously prepared by isc_dsql_prepare().
dialect	The SQL dialect to use for the statements in the batch update.
number_of_rows	The number of XSQLVARS for this batch update.

Argument	Description
insqlda	The address of an XSQLDA data structure describing the input parameters.
batch_vars	An array of pointers to XSQLVAR structures that describe each input parameter.
rows_affected	A pre-allocated array of type ULONG, which will be used to store the number of rows affected by the corresponding SQL statement in the sql array.

If a statement fails, the ISC_STATUS will be set, and the corresponding entry in the rows_affected array will be set to -1, or 0xFFFFFFFF.

To facilitate the allocation of the XSQLVAR array, a new macro is provided to calculate the size of the array. The new macro, XSQLVAR_LENGTH is defined as follows:

XSQLVAR_LENGTH(num_rows, num_vars_per_row)

Table 6.4 explains the meaning of each macro parameter.

Table 6.4 XSQLVAR_LENTGH Macro Parameters

Argument	Description
num_rows	The number of rows to be batched.
num_vars_per_row	The number of input parameters to be set per row.

Using the isc_dsql_batch_execute_immed Batch Update API

The following examples demonstrate how to use the isc_dsql_batch_execute_immed function. The code in Example 6.1 attaches to the *employee.gdb* database. Later, the example shows how to send a batch of two SQL commands: One INSERT statement, and one DELETE statement.

Example 6.1 Attaching to a database

isc_db_handle DB = NULL; /* Database handle */

```
isc_tr_handle trans = NULL /* Transaction handle */
long status[20]; /* Status vector */
char Db_name[128]; /* Database name */
char user_name[] = "example";
char password[] = "3xample";
char instance_name[256] = "gds_db1";
char dpb_buffer[256], *dpb, *p;

/* Set up the data base connection parameters */
dpb = dpb_buffer;
*dpb++ = isc_dpb_version1;
```

```
*dpb++ = isc_dpb_user_name;
*dpb++ = strlen(user_name);
for(p = user_name; *p;)
   *dpb++ = *p++;

*dpb++ = isc_dpb_password;
*dpb++ = strlen(password);
for(p = password; *p;)
   *dpb++ = *p++;

*dpb++ = isc_dpb_instance_name;
*dpb++ = strlen(instance_name);
for(p = instance_name; *p;)
   *dpb++ = *p++;

if(isc_attach_database(status, 0, Db_name, &DB, dpb_length, dpb_buffer))
   ERREXIT(status, 1);
```

Next, Example 6.2 demonstrates how to set up the buffers to hold the SQL statements.

Example 6.2 Preparing Buffers for a Batch Update

```
char *sql1 = "INSERT INTO DEPARTMENT (dept_no, department, head_dept) values
('117', 'Field Office: Hong Kong', '110')";

char *sql2 = "DELETE FROM DEPARTMENT WHERE dept_no = '117'";

char *sql_statements[2];
ULONG rows[2] = {0, 0};


sql_statements[0] = sql1;
sql_statements[1] = sql2;
```

Example 6.3 shows how to execute the batch update.

Example 6.3 Executing a Batch Update

```
/* Start a transaction */
if(isc_start_transaction(status, &trans, 1, &DB, 0, NULL))
   ERREXIT(status,1);

/* Submit the batch update */
if(isc_dsql_batch_execute_immed(status, &DB, &trans, 3, 2, sql_statements,
&rows))
   ERREXIT(status, 1);

/* Print results and end the transaction */
printf("Returned rows from the batch command: %d, %d", rows[0], rows[1]);
printf("Done with isc_dsql_execute_immed\n");
if(isc_commit_transaction(status, &trans))
   ERREXIT(status, 1);
```

The code above executes two statements in a batch update: One INSERT statement, and one DELETE statement. Notice the individual statements do not need to be terminated with a semicolon.

The number of rows affected by each statement is stored in the array called rows. The array must contain one element for each SQL statement executed in the batch update.

Using the isc_dsql_batch_execute API

The next group of examples show how to use the isc_dsql_batch_execute API to execute a parameterized INSERT statement. Whereas the isc_dsql_batch_exec_immed function sends a group of SQL statements, the isc_dsql_batch_execute function sends one parameterized statement with a group of values to use for the parameters.

First, establish the connection to the database, as shown in Example 6.1, “Attaching to a database”, above.

Example 6.4 demonstrates the declaration of variables in preparation for calling the isc_dsql_batch_execute function. The code will call the function using the two dept_no variables declared here.

Example 6.4 Declaring Variables for the isc_dsql_batch_execute Function

```
#define NUM_ROWS 2
#define NUM_VARS 2

//An UPDATE statement with 2 parameters..
char *sql1 = "UPDATE department SET budget = ? * budget + budget WHERE
dept_no = ?";

short flag0 = 0, flag1 = 0;
char dept_no[4] = "117", dept_no1[4] = "119";
isc_stmt_handle stmt_handle = NULL;
double percent_inc = (double)0.0;
int i;
XSQLVAR *array_sqlvar;
XSQLDA ISC_FAR *sqlda;
ULONG rows_affected[NUM_ROWS];
```

The code in Example 6.5 starts a transaction and prepares the parameterized UPDATE statement.

Example 6.5 Preparing the UPDATE statement

```
if(isc_start_transaction(status, &trans, 1, &DB, 0, NULL))
   ERREXIT(status, 1);

if(isc_dsql_allocate_statement(status, &DB, &stmt_handle))
   ERREXIT(status, 1);

sqlda = (XSQLDA ISC_FAR *)malloc(XSQLDA_LENGTH(2));
sqlda->sqln = 2;
sqlda->sqld = 2;
sqlda->version = SQLDA_CURRENT_VERSION;

if(isc_dsql_prepare(status, &trans, &stmt_handle, strlen(sql1), sql1, 3,
NULL))
   ERREXIT(status, 1);

array_sqlvar = (XSQLVAR *)malloc(XSQLVAR_LENGTH(sqlda->sqld, NUM_ROWS));
if(array_sqlvar == NULL)
   exit(-1);

// Prepare data for the parameters. The UPDATE statement takes two
parameters.
// This example batches two different values for the dept_no parameter.
array_sqlvar[0].sqldata = (char ISC_FAR *)&percent_inc;
```

```
array_sqlvar[0].sqltype = SQL_DOUBLE + 1;
array_sqlvar[0].sqllen = sizeof(percent_inc);
array_sqlvar[0].sqlind = &flag0;
flag0 = 0;

array_sqlvar[1].sqldata = dept_no;
array_sqlvar[1].sqltype = SQL_TEXT + 1;
array_sqlvar[1].sqllen = 3;
array_sqlvar[1].sqlind = &flag1;
flag1 = 0;

array_sqlvar[2].sqldata = (char ISC_FAR *)&percent_inc;
array_sqlvar[2].sqltype = SQL_DOUBLE + 1;
array_sqlvar[2].sqllen = sizeof(percent_inc);
array_sqlvar[2].sqlind = &flag0;
flag0 = 0;

array_sqlvar[3].sqldata = dept_no1;
array_sqlvar[3].sqltype = SQL_TEXT + 1;
array_sqlvar[3].sqllen = 3;
array_sqlvar[3].sqlind = &flag1;
flag1 = 0;
```

Finally, Example 6.6 calls the isc_dsql_batch_execute function and the results are printed.

Example 6.6 Execute the isc_dsql_batch_execute Function

```
if(isc_dsql_batch_execute(status, &trans, &stmt_handle, 3, sqlda, NUM_ROWS,
array_sqlvar, rows_affected))
   ERREXIT(status, 1);

for(i = 0; i < NUM_ROWS; i++)
   printf("After batch return values %d", rows_affected[i]);

free((void *)array_sqlvar);

if(isc_commit_transaction(status, &trans))
   ERREXIT(status, 1);

if(isc_detach_database(status, &DB))
   ERREXIT(status, 1);

free(sqlda);
```

ISQL Improvements

In ISQL, SQL statements to be executed in batch mode must be surrounded by the new BATCH START and BATCH EXEXCUTE commands. For example:

```
BATCH START;
...
(DDL/DML statements)
...
BATCH EXECUTE;
```

The BATCH EXECUTE command sends the statements between BATCH START and BATCH EXECUTE to the server. To begin another batch operation, you must issue another BATCH START command.

The following demonstrates a specific example of using batch mode with ISQL.

```
BATCH START;

INSERT INTO t1(f1, f2) VALUES (0,1);
UPDATE t1 SET f1=1 WHERE f2=1;

BATCH EXECUTE;
```

The first SQL statement in the example inserts a new row into table t1. The second statement updates the newly inserted row with a new value. Both of these statements are executed in one API call.

The AUTOCOMMITDDL mode of ISQL must be turned off in order to use batch updates.

Chapter

7

Database Settings

Database Write Mode Default SYNC

SYNC mode the default write mode for newly created databases in InterBase. This is changed from ASYNC to SYNC write mode. The call to write ASYNC mode buffers return immediately thus not guaranteeing that it has made to disk. Journaling now provides better performance for SYNC write databases, than ASYNC write mode. Use Journaling to get durability as well as performance, where needed.

Database File Preallocations

The InterBase SQL statement CREATE DATABASE now includes a preallocation clause to specify extra database space preallocated for the new database. The space is actually allocated when the user detaches from the connection that was established by the CREATE DATABASE statement. The database preallocation feature supports secondary database files in that the preallocation will be spread across all secondary files in accordance with their file size specifications.

Example

```
... [[NO] PREALLOCATE [=]  int  [PAGES]]
```

By default, creating a database does not preallocate additional database pages, so it is as if NO PREALLOCATE had been specified. This syntax is provided so that a DDL script can explicitly specify and document that preallocation has not been specified. Database preallocation is always specified in units of database pages to be consistent with other related features (i.e., length of secondary database files or shadow sets).

Important If a preallocation exceeds available disk space, the The IB thread making the write request when the device fills will timeout after 1 minute of waiting for the I/O to complete. Thereafter, it makes 4 additional I/O attempts, waiting 1 minute during each attempt, to complete the write (results written to the InterBase log). If space is not freed to allow the preallocation operation to continue the preallocation space requested will not be allocated.

Important

GSTAT

GSTAT display the database preallocation information, which is stored on the database header page. Following is a snippet from a GSTAT -H command:

Example

```
Variable header data:
    Preallocate pages: 5000
    Sweep interval: 25000
    *END*
```

ISQL Extract PREALLOCATE

The CREATE DATABASE command now includes the ISQL -extract PREALLOCATE clause to the formatted CREATE DATABASE statement if there is a non-zero preallocation value for the database. The ISQL *extract* operation can be invoked with the -a|-x options.

Example

```
/D/testbed>isql
Use CONNECT or CREATE DATABASE to specify a database
SQL> create database 'pr.ib' preallocate 500;
SQL> commit;
SQL> quit;
/D/testbed>ls -l pr.ib
-rwxrwxrwx 1 Administrators None 2048000 Jul 2 18:09
pr.ib /* It is 2MB size because each of the 500 database pages is
4KB in size */
/D/testbed>isql -a pr.ib

SET SQL DIALECT 3;


/* CREATE DATABASE 'pr.ib' PREALLOCATE 500
PAGE_SIZE 4096
*/


/* Grant permissions for this database */
/D/testbed>isql -x pr.ib


SET SQL DIALECT 3;


/* CREATE DATABASE 'pr.ib' PREALLOCATE 500 PAGE_SIZE 4096
*/
```

```
/* Grant permissions for this database */
/D/testbed>
```

GBAK

GBAK backs up and restores database preallocation information. This preallocation information will be silently ignored by earlier versions of the product that are not aware of the feature. A new switch has been added to GBAK to alter the stored preallocation in a database or backup file.

Switch -PR(EALLOCATE)

The switch -PR(EALLOCTE) takes an integer argument, which is the number of preallocation pages. This switch is legal for both backup and restore command-line options. For backup, the preallocation switch stores its argument in the backup file instead of the value specified in the database that is being backed up. For restore, the preallocation switch argument is used at the preallocation value in the restore database, instead of the value stored in the backup file. A GBAK preallocate switch value of 0 (zero) effectively disables database preallocation in the backup file or restored database. In GBAK verbose mode, database preallocation is logged to the console. The example below show a sample database backup in verbose mode. A similar message is logged for database restore.

Example

```
gbak -v foo.gdb foo.gbk -pr 5000
    ...
    gbak: readied database foo.gdb for backup
    gbak: creating file foo.gbk
    gbak: starting transaction
    gbak: database foo.gdb has a page size of 4096 bytes.
    gbak: database preallocation 5000 pages
```

If a database restore reduces the page size, the number of pages for database preallocation is automatically scaled upward to maintain a constant database preallocation size in bytes. If the restored page size is increased, database preallocation is reduced accordingly with a similar “Reducing” message written to the console. If the GBAK -PREALLOCATE switch was given then the automatic scaling of the database preallocation does not occur with changing page size. In other words, the -PREALLOCATE switch takes precedence.

Example

```
gbak -v foo.gdb foo.gbk -page_size 2048
...
      Reducing the database page size from 4096 bytes to 2048 bytes
      Increasing database preallocation from 5000 pages to 10000 pages
      created database foo1.gdb, page_size 2048 bytes
      database preallocation 10000 pages
```

API DPB Parameter

At the InterBase API-level, there is a new DPB parameter, isc_dpb_preallocate, that takes a 4-byte integer to specify database preallocation. It is only recognized and processed by isc_create_database(). isc_attach_database() silently ignores isc_dpb_preallocate.

With the InterBase service API, actions isc_action_svc_backup (isc_action_svc_restore) take new parameters, isc_svc_bkp_preallocate (isc_svc_rst_preallocate), respectively. Both parameters take a 4-byte argument to specify the database preallocation in units of database pages. The new service parameters have the same numeric value but two symbolic constants are provided for source code clarity to show the proper intent.

isc_db_preallocate Database Parameter

Finally, there is an isc_info_db_preallocate database info parameter to request database preallocate information stored on the database header page.

Chapter

8

Using BLOBs With VARCHAR Data

InterBase 2007 supports new SQL syntax that allows you to use BLOBs and VARCHAR data interchangeably.

This chapter describes the new SQL syntax that supports this functionality.

Text BLOBs and VARCHAR Data

All BLOB sub-types can be used interchangeably with VARCHAR data. However, with BLOB SUB_TYPE 1, the BLOB is considered to have a character type, essentially making the BLOB a CLOB data type. For BLOB columns of SUB_TYPE 1, the server converts character data to the column's character type before inserting, updating or comparing the data.

For all other sub-types, the BLOB data type accepts character input and treats it just as it would all other binary data. Hence, the BLOB data type treats all textual data as an array of bytes. Text data used in ISQL has a character set associated with it. This will most likely be the character encoding of the machine running ISQL (or any other client).

The server does not perform any character set conversion in these cases. Again,

the server treats the data as an array of bytes. To convert or store the textual data to a particular encoding (other than the system encoding), cast the character data to the required character set.

Text BLOB SQL Syntax

The general syntax for the SQL SELECT statement with a BLOB data type is:

```
SELECT CAST (<blob-column-name> as CHAR[<n>]) FROM <table-name>;
```

However, this feature allows text blobs to be interchangeable with VARCHAR data. So, it will improve InterBase SQL to allow new SQL syntax like ….

```
INSERT INTO <table-name> values (<text values>, ….);
UPDATE <table_name> set <blob column name> = <text value>;
```

And:

```
SELECT CAST (<blob column name> as CHAR[128]) from table;
SELECT * from <table name> where cast (<blob column> as VARCHAR[10]) =
"SMISTRY";
```

Also in addition, store procedures which accept a BLOB will be able to accept a text value as a parameter and implicitly be converted to a text blob.

For example:

```
CREATE PROCEDURE MYTEST (AINT INTEGER, INBLOB BLOB)
AS
Declare variable var_blob blob;
begin
insert
var_blob
```

This procedure can now be called using the following …

```
Execute procedure mytest (1, 'hello world');
```

Using Text BLOBs with VARCHAR Data

The SELECT CAST, UPDATE, and INSERT INTO statements can be used with the InterBase Client APIs. In this case, the values are returned as C structures. Specifically, the returned XSQLVARS would be of the type SQLVARYING, with the length of the text followed by the text data.

Example 8.1, "Using Text BLOBs"demonstrates the use of the new SQL syntax for text BLOBs.

Example 8.1 Using Text BLOBs

```
/* Same syntax to create a table... */
/* Note all sub-types are supported; SUB_TYPE 1 forces conversion */
/* to the column's character data type. */
CREATE TABLE BLOB_TEST (B_ID INT, BLOB_CL BLOB SUB_TYPE 1);
COMMIT;
/* New functionality for the INSERT statement... */
INSERT INTO BLOB_TEST VALUES (1, 'Fellowship of the Ring');
INSERT INTO BLOB_TEST VALUES (2, 'The Two Towers');
INSERT INTO BLOB_TEST VALUES (3, 'Return of the Jedi');
/* New syntax for UPDATE... */
UPDATE BLOB_TEST SET BLOB_CL='Return of the King' WHERE B_ID=3;
COMMIT;
/* New syntax for SELECT. The BLOB will be returned as a TEXT string. */
SELECT B_ID, CAST (BLOB_CL AS VARCHAR(25)) FROM BLOB_TEST;
```

The result of these statements in ISQL would be:

Table 8.1 Text BLOB Example Result

B_ID	BLOB_CL
1	Fellowship of the Ring
2	The Two Towers
3	Return of the King

Chapter

9

Internationalization Changes

Please note the following improvements in internationalization support for InterBase SP2.

Support for the UTF-8 Character Set

The UTF-8 character set is an alternative coded representation form for all of the characters of the ISO/IEC 10646 standard.

To use the UTF-8 character set, you would declare a database schema to use the character set, in the CREATE DATABASE SQL statement, as shown below:

CREATE DATABASE <filespec> <...> DEFAULT CHARACTER SET UTF8;

Additionally, you may use the alias UTF_8.

The attributes for the UTF-8 character set are shown in Table 6.1.

Table 9.1

Character Set	Character Set ID	Max Char Size	Min Char Size	Collation Orders
UTF8/UTF_8	59	1	4	N/A at this time

UNICODE_BE and UNICODE_LE Character Sets

InterBase now supports 16-bit UNICODE_BE and UNICODE_LE as server character sets. These character sets cannot be used as client character sets. If your client needs full UNICODE character support, please use UTF8 instead of UNICODE_LE and UNICODE_BE for the client character set (a.k.a LC_CSET). A client can use the UTF8 (or other native) client character set to connect with a UNICODE database.

A database schema is declared to use the new character set in the CREATE DATABASE statement, as follows:

CREATE DATABASE <filespec> <...> DEFAULT CHARACTER SET UNICODE;

Note that InterBase uses “big endian” ordering by default.

The attributes for the UNICODE_BE and UNICODE_LE character sets are shown in Table 6.2.

Table 9.2

Character Set	Character Set ID	Max Char Size	Min Char Size	Collation Orders
UNICODE_BE UCS2BE	8	2	2	N/A at this time
UNICODE_LE UCS2LE	64	2	2	N/A

Collations

InterBase 2007 does not support UNICODE collations in this release. The default collation is binary sort order for UNICODE.

PT_BR Collation For Brazilian Portuguese

New collations are declared to a database schema via the normal CREATE TABLE statement with the COLLATE clause:

Example

```
CREATE TABLE <table name> (<column name> <data type> COLLATE <COLLATION
NAME>);
```

Table 9.3 PT_BR Character/Collation Order

Character Set	Collation Order
ISO8859_1	CC_PTBRLAT1
ISO8859_15	CC_PTBRLAT9
WIN1252	CC_PTBRWIN

Note This collation is case and accent insensitive.

Note For more information see The Operations Guide provided with this release.

Chapter

10

UDF Descriptors

InterBase currently allows users to created User defined functions (UDFs), these UDFs are actually code written in C/C++ or Delphi and compiled to into a DLL. The UDFs are then defined with in the database using InterBase's DECLARE EXTERNAL FUNCTION command.

InterBase currently requires that parameters for these user defined functions be passed to the User defined function code from the database server as a value or by reference. In this case the any additional information is lost when the particular data type is converted to a native language supported data type like a “char *” or “int” or “short”. New functionality allows a particular parameter to be passed as a descriptor. When an argument is passed as a descriptor the InterBase server ensures all the information it has about the particular data type is passed to the function. In this way the structure can be probed to check if the values is a SQL NULL.

Further information regarding the character set can be obtained for textual data. Information regarding the precision and scale is available for numeric data.

Declaring a New UDF Using a Descriptor Parameter

The DECLARE EXTERNAL FUNCTION command has been improved to allow parameters to be passed as descriptors, the text in bold has been added to this existing command:

Example

```
  DECLARE EXTERNAL FUNCTION name [ datatype ;
  | CSTRING ( int ) | DESCRIPTOR  [, datatype | CSTRING ( int ) |
DESCRIPTOR  ]]
  RETURNS { datatype [BY VALUE] | CSTRING ( int ) | PARAMETER n } [FREE_IT]
  ENTRY_POINT 'entryname ' MODULE_NAME 'modulename ';
For Example the following declares a new UDF...
  DECLARE EXTERNAL FUNCTION DESC_ABS
  DESCRIPTOR
```

```
RETURNS DOUBLE PRECISION BY VALUE
ENTRY_POINT 'IB_UDF_abs' MODULE_NAME 'smistry_udf';
```

Note A parameter being passed as a descriptor cannot be used as a return type. This action will throw a error.

Defining the UDF:

The functions are defined in C/C++ or Delphi code. For C the developer needs to accept the descriptor parameter using the ISC_DSC structure. This structure is defined in the include file “ibase.h”. The above mentioned DESC_ABS function can be defined as follows in a C program file.

Example

```
double IB_UDF_abs (ISC_DSC *d)
  {
  double double_var ;
  /* function body */
  return double_var ;
  }
```

The ISC_DSC structure is defined as follows for C/C++ programs:

Example

```
  /**********************************/
  /* Descriptor control structure */
  /**********************************/
  typedef struct isc_dsc {
      unsigned char dsc_version; /* should be set to DSC_CURRENT_VERSION or
2 */
      unsigned char dsc_dtype;   /* the InterBase data type of this
particular parameter */
      char dsc_scale;            /* scale of the parameter for numeric data
types */
      char dsc_precision;        /* precision of the numeric data type */
      unsigned short dsc_length; /* size in bytes of the parameter */
      short dsc_sub_type;        /* for textual data types will have
information about character set and collation sequence,
                                    see DSC_GET_CHARSET and DSC_GET_COLLATE
macros for more information */
      unsigned short dsc_flags;  /* will be set to indicate null to
DSC_null or to DSC_no_subtype to indicate that
                                    the sub type is not set, this is a bit
map so multiple bits might be set,
                                    use binary operations to test, see
table below for explanation */
      unsigned char *dsc_address; /* pointer to the actual value of the
datatype */
  } ISC_DSC;
```

Some related macros follow:

```
  #define DSC_VERSION2 2
  #define DSC_CURRENT_VERSION DSC_VERSION2
  #define DSC_null 1
  #define DSC_no_subtype 2
  #define DSC_nullable 4
  #define dsc_ttype dsc_sub_type
  #define DSC_GET_CHARSET( dsc ) (( dsc )->dsc_ttype ) & 0x00FF)
  #define DSC_GET_COLLATE( dsc ) (( dsc )->dsc_ttype ) >> 8)
```

System Table Changes

The field RDB$MECHANISM in RDB$FUNCTION_ARGUMENTS system table has a value of “2” to indicate if the UDF argument is used as a Descriptor.

Chapter

11

Query Optimizer Improvements

The optimizer analyzes the tables and columns used in a given query and chooses indexes that speed up the searching, sorting, or joining operations.

Index Optimization of Correlated Subqueries in UPDATE statements

An indexed retrieval is now used to fetch rows from the correlated subquery in the **UPDATE** statement if there is an appropriate index defined. Utilize an indexed access path for correlated subqueries in **UPDATE** statements as in the following code example:

Example

```
UPDATE A SET A.C1 = (SELECT B.C1 FROM B WHERE B.C2 = A.C2)
```

Where **index** is **B.C2**, InterBase will use **index** to retrieve the matching row in table **B** where **B.C2 = A.C2**, since the row in the outer table **A** has already been fetched.

Shortcut Boolean Expression Evaluation

All general Boolean expressions involving **AND/OR** will be shortcut as soon as possible.

Note Re-ordering does note optimize the expressions.

Redundant Index Usage in Query Disjuncts

To avoid redundant indexes for disjuncts (**OR** conditions), query optimizer redundant indexes, not used to optimize conjuncts (AND conditions), now select one index that matches the most Boolean terms in the query. This greatly reduces the amount of index retrievals constructed.

Outer Join and Sort/Merge Optimization

The **Sort/Merge** for outer joins algorithm has been modified to recognize outer and inner streams of an outer join and match an outer row with a null-valued inner row when there is no matching row in the inner stream.

For full outer joins, the outer and inner streams are swapped after producing matching and null-matched rows for the first stream. The first stream becomes the inner stream and what was the second stream becomes the outer stream. These rows are then left outer joined and only those rows in which the outer stream is matched with nulls are produced. The matching rows on the join terms are filtered out because they were produced before the two streams were swapped during the first pass.

Invariant FALSE Restrictions in Queries

Query optimizer now looks for Booleans of the form **"literal <relop> literal"** that evaluate to **FALSE** and returns a false Boolean inversion node to short circuit data retrieval.

Chapter

12

JDBC URL Parameters

The feature enables a JDBC application to send in connection, data source and driver properties via the URL. Third party applications can now include run time parameters to the InterBase JDBC driver and the InterBase server, by appending them to the database URL property.

JDBC URL Argument

Use the JDBC URL argument for passing additional parameters as in the following example:

Example

```
String url =
"jdbc:interbase://localhost:3050/c:/dbs/books.ib?logWriterFile=logfile.txt"
;
```

Multiple properties can also be passed as:

Example

```
String url =
"jdbc:interbase://localhost:3050/c:/dbs/books.ib?logWriterFile=logfile.txt;
createDatabase=true”;
```

Legacy methods provide for by the Datasource and the DriverManager class are still retained and work as before, however note that the new functionality takes precedence over the Datasource and Drivermanager methods. Consider the following java code as an example:

Example

```
  {
  String url =
"jdbc:interbase://localhost:3050/c:/dbs/books.ib?logWriterFile=logfile.txt;
createDatabase=true";

  dataSource.setServerName ( "localhost");
  dataSource.setDatabaseName ( url );
  dataSource.setCreateDatabase ( false);
  }
```

In this case the create database flag in the URL will have precedence.

Log Writer File Property

A new property has been created which is only available via the database URL called logWriterFile, the usage is similar to other properties usage on the URL.

Example

```
?logWriterFile=c:/smistry/interclient.log
```

The setLogWriter call actually takes a defined PrintWriter, while the new logWriterFile takes a actual filename to be used as a logWriter.

Chapter

13

IBX Changes

InterBase Express (IBX) is a set of data access components that provide a means of building applications with the Borland Developer System (IDE for Delphi, C#, and C++) that can access, administer, monitor, and run the InterBase Services on InterBase databases.

This chapter describes changes to the InterBase Exchange software.

See the 'IBX' chapter of the Developer's Guide for more information about InterBase Express components.

See the Release Notes for information about using the IBX Update Kit.

Changes to IBX in InterBase 2007

The following elements of IBX have changed in InterBase 2007:

IBDatabase.pas - Added support for the IBDatabase param 'instance_name'. Use instance_name when you need to set a port to something other than the normal 3050.

IBDatabase.pas - Added support for the new Incremental Backup feature of InterBase 2007.

It takes two arrays as parameters. The first array is an array of file names. The second is an array of sizes. They should match up with each other. For instance, file array element one should match size element one. Only the final file is allowed to not have a size associated with it.

The final two parameters are Full and Overwrite. Passing in true for Full does a full backup, false creates an incremental backup (since the last dump).

Passing in true for Overwrite will overwrite existing files, while false will raise an error if the files exist.

IBQuery, IBTable, IBDataset, IBStoredProc - Added PSSetCommandText where WideStrings are passed.

IBScript - Added support for commit retaining and rollback retaining.

IBScript - Added support for the InterBase 2007 batch script APIs. Such as, in ISQL calling 'Batch start' starts a batch update and calling 'Batch Execute' will execute the batch statements. Only insert, update and delete statements are supported by this API.

Services - Added support for the instance_name parameter.

IBConfigService - Added two properties for database and transaction.

For the features added since 7.5, some of the properties are straight SQL DDL. In versions previous to InterBase 2007, IBConfigService would just create an INDatabase and INTransaction as needed.

InterBase will still create these as needed, but if you supply a database and transaction it will use those parameters first. This is needed for some features like the new journaling that requires exclusive access to the database.

TIBJournalInformation - New class for displaying and setting Journal settings on a database. This class is used to access the IBConfigService component properties:

HasJournal - Boolean, true if journaling is turned on.
HasArchive - Boolean, true if archiving has been turned on
CheckPointInterval - Integer
CheckPointLength - Integer
PageCache - integer
PageLength - Integer

PageSize - Integer
TimestampName - Boolean
Directory - String

IBConfigService - New property JournalInformation. Read and write to this property to manipulate journaling information on a database. Call GetJournalInformation to retrieve the Journaling information for a database.

a) CreateJournal - creates a journal based on the JournalInformation.
b) AlterJournal - alters a pre-existing journal system. Not all properties can be altered. See the Journaling chapter for limitations.
c) DropJournal - drops a journal system.
d) CreateJournalArchive - creates an archive. Takes an optional directory parameter.
e) DropJournalArchive - drops an archive.
f) GetJournalInformation - retrieves journaling information for this database and stores it in the JournalInformation property.
g) JournalInformation (property) - gives you access to the underlying IBJournalInformation field.

IBSecurityService - Fixed bug where modifying a users was actually trying to create a user.

IBVersionInfo - fixed a bug in the IsMinimumVersion function.

IBConfigService - Design time component editor now has a GetJournalInformation menu option. This service is only registered if you have InterClient 8.0 installed.

Chapter

14

IBConsole

InterBase provides an intuitive graphical user interface, called IBConsole, with which you can perform every task necessary to configure and maintain an InterBase server, to create and administer databases on the server, and to execute interactive SQL (ISQL).

This chapter describes changes to the Console for the InterBase 2007 server.

For more information about IBConsole, see the 'IBConsole' chapter of the Operations Guide.

Changes to IBConsole in InterBase 2007

Changes to IBConsole in InterBase 2007 include:

- You no longer have to login to use the Performance Monitor.
- The Tools menu now includes a 'Launch Licensing Manager' command that you can use to register a server.

Note If you are adding the very first certificate from IBConsole to an older database, the Add Certificate pop-up menu is shown. This will add an old style certificate. However, as of InterBase 2007, certificate information is not displayed, so it will be hidden, after the server version is determined.

Chapter

15

InterBase Features Per Release

This chapter reviews features introduced in InterBase 7.5, InterBase 7.1 and InterBase 7.0.

New in InterBase 7.5

InterBase 7.5 includes the following improvments:

Multi-Instance

InterBase 7.5 now allows multiple instances of InterBase servers to run simultaneously. In the past multiple versions of the InterBase server could not be run on the same machine. Previously when an application that utilized one version of InterBase, another application that utilized another version of InterBase could not be run. Now with InterBase 7.5 Borland has added the ability run multiple instances of InterBase on the same machine.

With InterBase 7.5 one previous version (major release) of InterBase, i.e. InterBase 7.1, or InterBase 6.x, etc. will be able to be run simultaneously. Multiple instances of InterBase 7.5 can be run simultaneously.

Note Separate licensing is required for all instances of InterBase.

Automatic re-routing of databases

Now that InterBase 7.5 allows multiple instances of InterBase to run on the same machine this feature will allow configurations where some database connections can be rerouted to a different InterBase server instance on the same machine. See the Operations Guide for implementation details.

Manual routing of databases

This solution allows application developers to explicitly specify a unique INTERBASE environmental variable for a local connection, or a unique TCP/IP protocol name for remote connections. This feature is useful if application developers want to isolate their application from other versions of InterBase installed on the same machine. See the Multi-Instance section in the Operations Guide for implementation details.

Server side database alias

Database alias renames a database file within the context of the server. This beneficial feature enables clients to connect to databases regardless of the knowledge of its exact location. See the Operations Guide for implementation details.

Embedded database user authentication

This is a security improvement new in InterBase 7.5. Now that InterBase 7.5 can manage multiple databases for unrelated applications the embedded database user authentication feature allows custom user account management that is not shared with other InterBase applications. See the Operations Guide for implementation details.

Note There is no embedded SQL support for DDL for embedded database user authentication.

New ODS

InterBase 7.5 uses ODS (On Disk Structure) 11.2. This new ODS is required to accommodate new system tables, changes to existing system tables, and embedded database user authentication. InterBase 7.0 ODS 11.0 databases and InterBase 7.1 ODS 11.1 databases are automatically upgraded to ODS 11.2 when an InterBase 7.5 server attaches to these databases. To migrate databases with an ODS less that 11.0 or 11.1, backup these database versions with the older version of InterBase and restore them using InterBase 7.5 IBConsole or **gbak**.

Note Since InterBase 7.5 automatically upgrades ODS 11.0 and 11.1 databases to ODS 11.2 it is recommended that you retain a copy of your database in an earlier ODS if you plan on using databases with InterBase 7.0 or 7.1.

Global temporary tables

This feature implements the functionality of SQL global temporary tables in InterBase 7.5. Previously InterBase developers simulated temporary tables with permanent base tables. The developer was responsible for the application dropping those tables and performing any housekeeping to empty those tables if the application or database server abnormally terminated. With this new SQL temporary table feature in InterBase 7.5 all of the namespace and life cycle issues are transparently managed once the temporary table is declared to the database schema, thus making application development much easier. See the Language Reference for implementation details.

Note There is no embedded SQL support for DDL on temporary tables.

CASE, COALESCE, and NULLIF

New language features have been added in InterBase 7.5; CASE, COALESCE, and NULLIF. SQL dialect 1-3 applications should be able to use this functionality except if the evaluation of a value expression yields a value of a data type only supported by SQL dialect 3. An InterBase server should raise an SQL dialect exception in this case. For syntax of these new language implementations, please refer to the Language Reference.

Note There is no embedded SQL support for CASE, COALESCE and NULLIF.

Index optimization for NULL/non-NULL values

Nulls are sorted high in indices, meaning that they are located at the end of the index. When a query with greater than is matched to that index, previous versions of InterBase would unnecessarily go out and retrieve all the null-valued record versions from the index even though there is no way the nulls will satisfy the query. This will optimize such queries by not involving the NULL key values where they are not required. This will improve the overall performance of the InterBase server, and give a better response to many SQL queries. Having NULL values in index

keys is quite common (more so in composite keys), and hence this will have a wider impact on all InterBase customers who will see better performance from the product.

Memory management allocation algorithms

In InterBase 7.5 the memory management allocation algorithms have been improved. In previous versions of InterBase the memory management allocation algorithms were designed for the Classic single process architecture. Those outdated algorithms constrained memory resources in the Super Server architecture; the best-fit search algorithm for heap memory allocation will be changed to a first-fit algorithm. Additionally, separate memory heaps will be managed for ad-hoc memory allocation and block-based demands from the SQL and relational engine components to prevent memory fragmentation. This will also enable a degree of SMP parallelization by allowing simultaneous memory allocations. The first-fit algorithm will minimize search latency due to soft/hard page faulting resulting from searching the entire server address space for free memory.

Stored procedure and trigger cache management

The cache management for stored procedures and triggers in InterBase 7.5 has been refined to prevent the server from locking large amounts of memory to maintain the residency of triggers and stored procedures. The cache management will operate on two levels: 1) Deallocate individual clones not in use, and 2) deallocate the primary requests which are not used.

Sort buffer cache management

This feature in InterBase 7.5 improves the page faulting that occurs when 1MB sort buffers are immediately released. This phenomenon can be observed in rapidly executing statements or procedures which perform operations requiring a sort buffer, or during the index build phase of database restoration.

Greater SMP scalability

This feature in InterBase 7.5 improves performance for higher numbers of CPU both physical and logical. Through the use of spin locks and thread-private latch caches the InterBase ATOM synchronization architecture reduces the number and frequency of synchronization points.

Database page buffer cache

In InterBase 7.5 the largest page size has been increased from 8,192 to 16, 384 bytes. The largest buffer cache has been increased from 65,000 to 131,000 pages.

Thread-private Latch Cache

The Thread-private Latch Cache (TLC) reduces the number and frequency of atom synchronization locks needed to safely address and pin down resident pages in the database page buffer cache. In doing so, performance is improved because of a shorter code path and the avoidance of thread context-switching due to contention over atom synchronization variables.

TLC improves InterBase performance on SMP servers and single CPU servers when the IBCONFIG parameter MAX_THREADS is not equal to 1. That is, when multiple threads are allowed to execute concurrently inside the InterBase engine. If MAX_THREADS is equal to 1 then atom synchronization and TLC are both disabled.

Error reporting improved in interbase.log

With improved error reporting in interbase.log database names are no longer reported in the 8.3 format. Index names instead of numbers are now reported when verifying a database, and index errors are reported. Also, improved errors are reported when errors are found in ib_license.dat and the InterBase server is started.

New in InterClient 4.7

InterClient 4.7 includes the following updates:

Savepoints

Savepoints were introduced to InterBase 7.1, InterClient 4.7 now surfaces this as a JDBC standard implementation with the class IBSavepoint

The following example shows how to use IBSavepoint:

```
package bo_blob;
import java.sql.*;

public class testSavepoints
{
  public static void main(String[] args)
  {
    try
    {
      System.out.println("InterClient version: " +

interbase.interclient.Driver.getInterClientVersionInfo());

      String url = "jdbc:interbase://localhost/c:\\smistry\\foo.ib";
      Connection con;
      Class.forName("interbase.interclient.Driver");
      con = DriverManager.getConnection(url, "sysdba", "masterkey");

      con.setAutoCommit(false);
      Statement stmt = con.createStatement();
```

```
      try
      {
        stmt.executeUpdate("create table test (a int)");
      }
      catch (SQLException se)
      {
        System.out.println(se.getMessage());
      }
      con.commit();
      Savepoint savepoint = con.setSavepoint("sean");
      Savepoint savepoint = con.setSavepoint();
      PreparedStatement pstmt = con.prepareStatement("insert into test
values (?)");
      stmt.executeUpdate("insert into test values (1)");
      ResultSet rs = stmt.executeQuery("Select a from test");
      while (rs.next())
      {
        System.out.println("a = " + rs.getInt("a"));
      }
     con.rollback(savepoint);
      rs = stmt.executeQuery("Select a from test");
      while (rs.next())
      {
        System.out.println("a = " + rs.getInt("a"));
      }
      con.commit();
      stmt.executeUpdate("drop table test");
    }
    catch (Exception se)
    {
      se.printStackTrace();
    }
  }
  public testSavepoints()
  {
  }
}
```

ParameterMetaData

InterClient had always implemented a version of ParameterMetaData, as it was not specified by JDBC, it was a InterClient extension to the specification. With JDK 1.4 this is now a part of JDBC 3.0 specification and InterClient surfaces this as a JDBC call. In order to use this new JDBC specified class users will need to use java.sql.ParameterMetaData, the previous class interbase.interclient.ParameterMetaData is the older implementation and is now deprecated.

Note The usability of these 2 new classes is based on the JDBC 3.0 specification and visiting Sun's JDBC 3.0 web site for detailed information on what these new interfaces include is recommended.

New in InterBase 7.1

InterBase 7.1 includes the following updates:

New cross-platform installer

InterBase 7.1 has a new all-Java installer that is available for all InterBase platforms. For information about installation, please see the *IBsetup.html* file located at the root of your InterBase CD-ROM or download file. This file is also accessible from the main screen of the installer.

New registration

InterBase now joins other Borland products in requiring product registration. As part of the install process, you are asked to register and will be given four options for doing this, discussed below.

Preparing for the install and registration If you have already completed the InterBase installation and registration, you can skip this section. Before you begin the installation process have the following handy:

- The Serial Number and Key that are provided on the jacket of your InterBase CD-ROM.
- Your Borland Developer Network membership information. You can supply either your BDN user name or the E-mail address that you used to sign up plus the password. If you have ever registered a Borland product, you are a member of the BDN network. In addition, you might have signed up for membership on one of the Borland web sites. If you are not a member of the Borland Developer Network, you have an opportunity to join during the registration process.

IBsetup.html provides detailed information about the installation and registration process.

Precision of exact numerics

InterBase now returns the precision of exact numeric data types back to the client using the XSQLDA structure.

New drivers

For Windows platforms, InterBase 7.1 includes a Borland Data Provider (BDP) for ADO.NET programming.

Old drivers The IBX drivers for Delphi 5, Delphi 6, and C++Builder 5 are still available on your InterBase 7.1 CD-ROM, but they are no longer listed on the driver install menu.

New ODS

InterBase 7.1 uses ODS 11.1 rather than the ODS 11.0 used by InterBase 7.0. This new ODS is required to accommodate reporting the precision of exact numerics. To migrate databases, back them up with the older version of InterBase and restore them using InterBase 7.1 IBConsole or **gbak**.

Savepoints

InterBase 7.1 implements savepoints as defined in the SQL 1999 standard.

Savepoints in SQL

In DSQL and ESQL the following SQL statements are available:

1. To create a savepoint:

   SAVEPOINT <*savepoint_name*>

   A savepoint name can be any valid SQL identifier. Savepoint names must be unique within their atomic execution context. If you assign a name that is already in use, the existing savepoint is released and the name is applied to the current savepoint. An application, for example, is an execution context, as is each trigger and stored procedure. Thus, if you have an application with several triggers, you can have a savepoint named SV1 within the application and also within each trigger and stored procedure.

2. To release a savepoint:

   RELEASE SAVEPOINT <*savepoint_name*>

   Releasing a savepoint destroys that savepoint without affecting any work that has been performed subsequent to its creation.

3. To roll back to a savepoint:

   ROLLBACK [WORK] [TO SAVEPOINT <*savepoint_name*>]

   Issuing a ROLLBACK TO SAVEPOINT command rolls back all work performed since the creation of the named savepoint. If other savepoints were created after the named savepoint, those later savepoints are also rolled back.

Savepoints in the InterBase API

The InterBase API supports savepoints with the following functions:

1. To create a savepoint:

```
ISC_STATUS isc_start_savepoint(
  ISC_STATUS *status_vector,
  isc_tr_handle *trans_handle,
  char *savepoint_name);
```

2. To release a savepoint:

```
ISC_STATUS isc_release_savepoint(
```

```
    ISC_STATUS *status_vector,
    isc_tr_handle *trans_handle,
    char *savepoint_name);
```

3 To roll back to a savepoint:

```
ISC_STATUS isc_rollback_savepoint(
    ISC_STATUS *status_vector,
    isc_tr_handle *trans_handle,
    char *savepoint_name
    short option);
```

The *option* parameter is reserved for future use. Pass a value of zero for this parameter.

Savepoints in triggers and stored procedures

Savepoints are implemented in stored procedures and triggers.

A SAVEPOINT example

The following code snippet is a simple example of how to use savepoints:

```
CREATE PROCEDURE add_emp_proj2 (emp_no SMALLINT, emp_name VARCHAR(20),
proj_id CHAR(5))  AS
BEGIN
 BEGIN
   SAVEPOINT EMP_PROJ_INSERT;
   INSERT INTO employee_project (emp_no, proj_id) VALUES
(:emp_no,:proj_id);
   WHEN SQLCODE -530 DO
   BEGIN
       ROLLBACK TO SAVEPOINT EMP_PROJ_INSERT;
       EXCEPTION unknown_emp_id;
   END
 END
SUSPEND;
END;
```

New keywords

The savepoint functionality adds the following new keywords:

SAVEPOINT RELEASE

Performance monitoring now accessible in IBConsole

You can now access the performance monitoring features that were introduced in InterBase 7.0 through IBConsole, the graphical Windows interface for InterBase.

New character sets

InterBase 7.1 implements several new character sets and collation orders.

For the Latin 2 character set, InterBase implements Polish and Czech. More languages will be implemented in the future. A longish list of languages is implemented for Latin 9, listed in the table below. Finally, Russian is implemented for the KOI8-R character set.

These new character sets are defined as follows:

Character set	Char. set ID	Max. char. size	Min. char. size	Collation orders
ISO8859_2 (Latin2)	22	1 byte	1 byte	ISO8859_2 CS_CZ PL_PL
ISO8859_15 (Latin9)	39	1 byte	1 byte	ISO8859_15 DA_DA9 DE_DE9 DU_NL9 EN_UK9 EN_US9 ES_ES9 FI_FI9 FR_CA9 FR_FR9 IS_IS9 IT_IT9 NO_NO9 PT_PT9 SV_SV9
KOI8-R	58	1 byte	1 byte	RU_RU

Note Databases can optionally have a default character set defined for them. Character sets can also optionally be defined for specific table columns. If you are connecting to a database from a platform whose default code page is different from that of the database you are connecting to, you must specify the default code page of the client platform when making the connection to the database. To do this from IBConsole, select the database from the Tree Pane and choose Connect As from the Connect menu or the mouse context menu. The resulting Database Connect dialog box contains a Character Set field where you can specify the client platform character set from the pull-down list.

Improved SMP support

Support for multiprocessor machines has been improved. Among other changes, the MAX_THREADS parameter in the *ibconfig* configuration file now defaults to 1,000,000 when two or more CPUs are present and licensed. This means that there are never threads waiting to execute and improves the speed with which they release any resources that they hold. When only one CPU is licensed or if only one CPU is present, MAX_THREADS defaults to 1. For the purpose of

determining this default value, InterBase counts a hyper-threaded processor as a single CPU. You can change the number of simultaneous active server threads by editing the MAX_THREADS entry in the *ibconfig* configuration file.

Hyper-threading support on Intel processors

InterBase can support hyper-threading on Intel processors that support logical processors using Intel's hyper-threading technology. To enable this support in the InterBase server, you must make a setting in the InterBase configuration file, *ibconfig*. If you are running the InterBase server on a machine with hyper-threaded processors, edit the ENABLE_HYPERTHREADING parameter in the configuration file. By default, this parameter is set to zero. Set the value to 1 to allow the InterBase server to use hyperthreaded processors.

Change in gbak functionality

When restoring a database, **gbak** no longer automatically performs constraint checking the database during the restore process. This improves the speed of database restores and ensures that users can always restore their databases from backup files even when the backup files contain data that violates constraints such as NOT NULL, CHECK, PRIMARY and UNIQUE indexes, or REFERENTIAL constraints.

InterBase 7.1 provides now switches and parameters to provide the former capability of validating a database when restoring it.

Command line	There is a new command-line switch: -VA[LIDATE]. For example: gbak -r -user joe -pass blurf@ C:\archive\foo.ibk jupiter:/foo.ib -validate
InterBase Services API	There is a new parameter that enables validation during a restore: *isc_spb_res_validate*.
DPB	There is a new DPB parameter, *isc_dpb_gbak_validate* that instructs the server to include validation checks during a database restore.

Hyper-threading support for Intel processors

InterBase now recognizes and responds to hyperthreading technology in Intel processors. In InterBase 7.0, there was no way to exploit hyperthreading without purchasing additional SMP licenses. InterBase 7.1 now unlocks the additional processing power of hyperthreading transparently, without requiring additional SMP licenses. These same changes also insure that an SMP license is applied to a physical processor and not a logical processor, for maximum price/performance benefit.

New SQL command: DROP GENERATOR

InterBase now supports a DROP GENERATOR SQL statement:

DROP GENERATOR *generator_name*

The statement fails if *generator_name* is not the name of a generator defined on the database. This command checks for any existing dependencies on the generator—for instance in triggers or UDFs—and fails if such dependencies exist. An application that tries to call a deleted generator returns runtime errors.

DROP GENERATOR is implemented for DSQL and **isql**.

In previous versions of InterBase that lacked the DROP GENERATOR command, users were told to issue a SQL statement to delete the generator from the appropriate system table. This approach is strongly discouraged now that the DROP GENERATOR command is available, since modifying system tables always carries with it the possibility of rendering the entire database unusable as a result of even a slight error or miscalculation.

Improved garbage collection/index handling

Users will see significant performance improvement as a result of InterBase 7.1's more efficient garbage collection of duplicate index nodes. New algorithms have been added that minimize computational overhead and memory consumption during garbage collection.

IBConsole displays additional object dependencies

IBConsole now displays object dependencies on generators and UDFs in addition to all the dependencies it formerly displayed.

Using the InterBase Install API

Certain components of the InterBase Install API point to an InterBase file structure that is no longer in use. If you are writing or updating an install application using this API, you need to have the current InterBase files arranged in the file structure required by the InterBase Install API. To facilitate this, InterBase supplies a file, *silent_install.zip*, that contains all the current files arranged in the structure required by the API.

If you are writing an install application, extract *silent_install.zip*, and place your compiled install application at the root of the resulting file structure.

If you have an existing install application that does not bundle InterBase files within the binary, you can update it by just extracting *silent_install.zip*, and copying the resulting files over the InterBase file structure that you previously used.

If your existing install application includes changed InterBase files within the binary, you need to refresh the file structure with the files in *silent_install.zip*, and then recompile the application.

Licensing: VARs are now provided with an additional file, which must be included with their InterBase installs in order to provide valid product registration. If you are embedding or reselling InterBase, you should have received instructions about how to manage this file. If you need more information, contact your Borland InterBase representative.

Documentation fixes and changes

Some errors have been corrected in the documentation for InterBase 7.1. These changes are included in the PDF documents that ship with InterBase 7.1. They are not yet included in the printed documents.

UDF library documentation has been moved

In order to make all UDF information available in one place, the UDF chapter has been removed from the *Language Reference* and folded into the "Working with UDFs and Blob Filters" chapter of the *Developer's Guide*.

Declaring BLOB UDFs

The documentation now contains a more complete description of how to declare a UDF that returns a Blob.

To specify that a UDF should return a BLOB, use the RETURNS PARAMETER *n* statement to specify which input Blob is to be returned. For example, if the BLOB to be returned is the third input parameter, specify RETURNS PARAMETER 3. The Blob_PLUS_Blob UDF concatenates two BLOB and returns the concatenation in a third BLOB. The following statement declares this UDF to a database, specifying that the third input parameter is the one that should be returned:

```
DECLARE EXTERNAL FUNCTION Blob_PLUS_Blob
   Blob,
   Blob,
   Blob
   RETURNS PARAMETER 3
   ENTRY_POINT 'blob_concatenate' MODULE_NAME 'ib_udf';
COMMIT;
```

For more information about UDFs and Blobs, see the chapter "Working with UDFs and Blob Filters" in the *Developer's Guide*.

Calling convention for UDFs

Previous versions of InterBase documentation said that UDFs should be called using _stdcall. This is not correct. InterBase uses the CDECL calling convention, so all UDFs must be declared using the CDECL calling convention.

Portable UDFs

It has always been the case that UDFs could be written for Unix and Linux platforms as well as for Windows. However, the documentation did not make this clear. This has now been corrected. In addition, examples of declaring UDFs now show the portable form, in which the extension of the module name is not included. For example:

```
DECLARE EXTERNAL FUNCTION LOWERS VARCHAR(256)
   RETURNS CSTRING(256) FREE_IT
   ENTRY POINT 'fn_lower' MODULE_NAME 'udflib';
```

Correction for YEARDAY range

In the *Language Reference*, the range for EXTRACT(YEARDAY) should be 0–365. This will be corrected in the next version of the InterBase document set. It is incorrect in the set that ships with InterBase 7.1.

New in InterClient 4.0

Data Source properties for InterBase

Standard properties

Table 15.1 *Data Source* standard properties

Name	Type	Description	Default Value
databaseName	String	The name of the database to connect to	null
serverName	String	The InterBase server name	localhost
user	String	The InterBase user who is connecting	null
password	String	The InterBase user password	null
networkProtocol	String	The InterBase network protocol; this can only be jdbc:interbase: for InterClient.	jdbc:interbase
portNumber	int	The InterBase port number	3050
roleName	String	The InterBase role	null
dataSourceName	String	The logical name for the underlying XADataSource or Connection Pool; used only when pooling connections for InterBase (XA is not supported)	null
description	String	A description of this data source	null

Extended properties

Table 15.2 *Data Source* Extended properties

Name	Type	Description	Default Value
charSet	String	Specifies the character encoding for the connection; used for sending all SQL and character input data to the database and for all output data and InterBase messages retrieved from the database. The encoding specified by *charSet* must match one of the supported IANA character-encoding names detailed in the *CharacterEncodings* class. If *charSet* is set to *NONE*, InterClient uses the default system encoding obtained by the *System.getProperty("file.encoding")* method if that default encoding is supported by InterBase. If the default system encoding is not supported by InterBase, it is recommended that you use the *charSet* property to set the InterClient *charSet* to one of the InterBase-supported encodings. InterClient messages do not utilize *charSet*, but derive from the resource bundle in use, which is based on the locale-specific encoding of the client.	No default value
sqlDialect	int	The client SQL dialect. If the value is set to 0 then the database's dialect is used for the client dialect.	0
createDatabase	Boolean	If set, the database is created if it does not exist.	false
serverManagerHost	String	Ignored.	null
sweepOnConnect	boolean	If set, forces garbage collection of outdated record versions immediately upon connection See the InterBase *Operations Guide* for more details. Sweep does not require exclusive access, but there is some data and transaction state information that can be updated only where there are no active transactions on the database.	false

Table 15.2 *Data Source* Extended properties (*continued*)

Name	Type	Description	Default Value
suggestedCachePages	int	The suggested number of cache page buffers to use for this connection This is a transient property of the connection and is overridden by the database-wide default set by *ServerManager.setDatabaseCachePages(database, pages)*. It takes precedence over the server-wide default set by DATABASE_CACHE_PAGES in the InterBase *ibconfig* startup file or by *ServerManager.startInterBase(defaultCachePages, defaultPageSize)*. On SuperServer, if a database cache already exists due to another attachment to the database, then the cache size can be increased but not decreased. So, although this is a transient property, once the cache size is increased, it stays that way as long as there are active connections. Once all connections to the database are closed, then subsequent connections use the database-wide or server-wide defaults. Note: Using this connection property can jeopardize the performance of the server because an arbitrary user can connect and reserve 200MB for *foo.ib* while *corporate.ib* is forced to accept less. InterBase code sets an absolute limitation on MAX_PAGE_BUFFERS of 65,535 pages. So the cache memory size for a database cannot go beyond a maximum of MAX_PAGE_BUFFERS**PageSize* bytes, which is 512MB for an 8K page size. 8K is the maximum database page size currently allowed. If this property is zero or unspecified and there is no server-wide or database-wide default set, the default pages used is 2048 cache pages. Also see *DatabaseMetaData.getPersistentDatabaseCachePages()*, and *DatabaseMetaData.getActualCachePagesInUse()*.	0

InterClient connection pooling

InterClient now works with Container Managed Persistence (CMP) 2.0, which is supplied with the Borland Enterprise Server. This enables JDBC DataSource 2.x connectivity to InterBase databases. The following *jndi-definition.xml* file shows how it can be used through an application server:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jndi-definitions PUBLIC "-//Borland Corporation//DTD
JndiDefinitions//EN"
"http://www.borland.com/devsupport/appserver/dtds/jndi-definitions.dtd">
<jndi-definitions>
    <visitransact-datasource>
        <jndi-name>serial://datasources/DataSource</jndi-name>

<driver-datasource-jndiname>serial://datasources/driverDataSource</driver-d
atasource-jndiname>
        <property>
            <prop-name>connectionType</prop-name>
            <prop-type>Enumerated</prop-type>
```

```
            <prop-value>Direct</prop-value>
        </property>
        <property>
            <prop-name>dialect</prop-name>
            <prop-type>Enumerated</prop-type>
            <prop-value>interbase</prop-value>
        </property>
    </visitransact-datasource>
    <driver-datasource>
        <jndi-name>serial://datasources/driverDataSource</jndi-name>

<datasource-class-name>interbase.interclient.JdbcConnectionFactory</datasou
rce-class-name>
        <property>
            <prop-name>user</prop-name>
            <prop-type>String</prop-type>
            <prop-value>SYSDBA</prop-value>
        </property>
        <property>
            <prop-name>password</prop-name>
            <prop-type>String</prop-type>
            <prop-value>masterkey</prop-value>
        </property>
        <property>
            <prop-name>serverName</prop-name>
            <prop-type>String</prop-type>
            <prop-value>agni</prop-value>
        </property>
        <property>
            <prop-name>databaseName</prop-name>
            <prop-type>String</prop-type>
            <prop-value>c:/admin.ib</prop-value>
        </property>
       <property>
            <prop-name>sqlDialect</prop-name>
            <prop-type>int</prop-type>
            <prop-value>3</prop-value>
        </property>
        <property>
            <prop-name>createDatabase</prop-name>
            <prop-type>boolean</prop-type>
            <prop-value>true</prop-value>
        </property>

    </driver-datasource>
</jndi-definitions>
```

InterClient scroll ability

The Connection class

To achieve JDBC 2.0 core compliance, InterClient now allows a value of *TYPE_SCROLL_INSENSITIVE* for the *resultSetType* argument for the following *Connection* methods:

public java.sql.Statement createStatement (int resultSetType, int resultSetConcurrency)

public java.sql.CallableStatement prepareCall (String sql, int resultSetType, int resultSetConcurrency)

public java.sql.PreparedStatement prepareStatement (String sql, int resultSetType, int resultSetConcurrency)

Previously, the only allowable value for *resultSetType* was *TYPE_FORWARD_ONLY.* Currently, the only type not allowed is the *TYPE_SCROLL_SENSITIVE*

The ResultSet class

The resultSetType property of the ResultSet class can now have a value of *TYPE_SCROLL_INSENSITIVE* Previously, the only allowable value for *resultSetType* was *TYPE_FORWARD_ONLY.* Currently, the only type not allowed is the *TYPE_SCROLL_SENSITIVE*.

The following methods now return a valid value when the resultSets that are of the new *resultSetType.TYPE_SCROLL_INSENSITIVE*:

public boolean isBeforeFirst()

public boolean isAfterLast()

public boolean isFirst()

public isLast()

public void beforeFirst()

public void afterLast()

public boolean first()

public boolean last()

public int getRow()

public boolean absolute(int row)

public boolean relative(int rows)

public boolean previous()

New InterClient methods

InterClient is InterBase's JDBC driver. In InterBase 7.0, we introduced a new type 4 JDBC driver. For InterBase 7.1, we have added a large collection of methods to this driver to bring it into compliance with the JDBC 2.0 standard.

Methods for the Statement and PreparedStatement classes

The following methods have been added to both the *Statement* and the *PreparedStatement* classes. The methods listed below now work according to the JDBC specifications.

Table 15.3 Methods for the *Statement* and *PreparedStatement* classes

Method	Functionality
`void Statement.addBatch(String sql)`	Adds sql to the current list of commands.
`void Statement.clearBatch()`	Empties the list of commands for the current statement object.
int[] *Statement.executeBatch()* throws BatchUpdateException	Submits the list of commands for this statement's objects to the database for execution as a unit. The returned integer array contains the update counts for each of the SQL commands in the list.
`void PreparedStatement.addBatch()`	Adds a set of parameters to the list of commands for the current *PreparedStatement* object's list of commands to be sent to the database for execution.

The BatchUpdateException class

A new *BatchUpdateException* class has been implemented in order to support JDBC Batch update functionality. Here is the list of methods and constructors in the new class:

Table 15.4 Methods and constructors for the new *BatchUpdateException* class

Method/Constructor	Functionality
public BatchUpdateException(*String reason,* *String SQLState,* *int vendorCode,* *int [] updateCounts)*	Constructs a *BatchUpdateException* object where: • *reason* is a string describing the exception, • *SQLState* is an object containing Open Group code identification, • *vendorCode* identifies the vendor-specific database error code • *updateCounts* contains an array of INTs where each element indicates the row count for each SQL UPDATE command that executed successfully before the exception was thrown.
public BatchUpdateException(*String reason,* *String SQLState,* *int [] updateCounts)*	Constructs a *BatchUpdateException* object where: • *reason* is a string describing the exception, • *SQLState* is an object containing the InterBase error code • *updateCounts* contains an array of INTs where each element indicates the row count for each SQL UPDATE command that executed successfully before the exception was thrown. • The vendor code is implicitly set to zero.
public BatchUpdateException(*String reason,* *int [] updateCounts)*	Constructs a *BatchUpdateException* object where: • *reason* is a string describing the exception, • *updateCounts* contains an array of INTs where each element indicates the row count for each SQL UPDATE command that executed successfully before the exception was thrown. • The following values are implicitly set: the *vendorCode* is set to zero and the Open Group code identification is set to null.
public BatchUpdateException(int [] updateCounts)	• Constructs a *BatchUpdateException* object where *updateCounts* contains an array of INTs in which each element indicates the row count for each SQL UPDATE command that executed successfully before the exception was thrown. • The following values are implicitly set: *reason* is set to null, *vendorCode* is set to zero, and the Open Group code identification is set to null.

Table 15.4 Methods and constructors for the new *BatchUpdateException* class (*continued*)

Method/Constructor	Functionality
public BatchUpdateException()	The following values are implicitly set: • *updateCounts* is set to a zero-length integer array, • *reason* is set to null, • *vendorCode* is set to zero, • the Open Group code identification is set to null.
public int [] *getUpdateCounts*()	Retrieves an array of INTs where each element indicates the row count for each SQL UPDATE command that executed successfully before the exception was thrown

The DatabaseMetaData.supportsBatchUpdates function

The *DatabaseMetaData.supportsBatchUpdates* function has changed as follows:

Function	Functionality
boolean DatabaseMetaData.supportsBatchUpdates()	Can now return TRUE.

Additional functions

Additional functions that implement the JDBC 2.x API functionality are listed below.

Function	Functionality
int Statement.getResultSetType()	Returns the type if *resultSet* is open, otherwise throws an exception
int Statement. getResultSetConcurreny()	Returns the concurrency if *resultSet* is open.
int Statement. getFetchDirection()	Returns the fetch direction if *resultSet* is open, the return value is always FETCH_FORWARD for InterBase.
int ResultSet. getFetchDirection()	Returns FETCH_FORWARD in all cases

Function	Functionality
int ResultSet. getFetchSize()	Returns the fetch size for the statement's result set.
int ResultSet. setFetchSize()	Allows you to set the fetch size of the resultset and the statement.
int ResultSet. setFetchDirection()	Throws an exception; it can only work with TYPE_SCROLL_SENSITIVE and TYPE_SCROLL_INSENSITIVE. Neither of these are supported by InterBase, since InterBase does not support scrollable cursors. The only ResultSet type allowed by InterClient/InterBase is TYPE_FORWARD_ONLY.

Code examples

Code example for the batch update functions:

```
Statement Class
con.setAutoCommit(false);
Statement stmt = con.createStatement();
stmt.addBatch("INSERT INTO foo VALUES (1, 10)");
stmt.addBatch("INSERT INTO foo VALUES (2, 21)");
int[] updateCounts = pstmt.executeBatch();
con.commit();
```

Code example for the *PreparedStatement* class:

```
PreparedStatement pstmt = con.prepareStatement ("UPDATE employee set emp_id
= ? where emp_id = ?")
pstmt.setInt(1, newEmpId1);
pstmt.setInt(2, oldEmpId1);
pstmt.addBatch();
pstmt.setInt(1, newEmpId2);
pstmt.setInt(2, oldEmpId2);
pstmt.addBatch();
int[] updateCounts = pstmt.executeBatch();
```

Code example for the *BatchUpdateException* class and *getUpdateCounts*() method

```
try
{
   int[] updateCounts = pstmt.executeBatch();
}
catch (BatchUpdateException b)
{
   int [] updates = b.getUpdateCounts();
   for (int i = 0; i < updates.length; i++)
   {
      System.err.println ("Update Count " + updates[i]);
   }
}
```

InterClient and the Borland Enterprise Server

InterClient 4 now works with Container Managed Persistence (CMP) 2.0, which is provided with Borland Enterprise Server (BES) 5.x.

Other InterClient Improvements

- The JDBC Timestamp data type now matches the InterBase SQL TIMESTAMP data type and allows fractions of seconds.
- The *Resources_ru.class* has been removed from the *interclient.jar* file to improve code page flexibility on non-Windows machines in Russian character sets.

New in InterBase 7.0

As a reminder, or for those of you who may have missed the InterBase 7.0 release, the following is a list of features that were new in InterBase 7.0, with a brief description of each.

- Database naming

  InterBase no longer recommends using ".gdb" as the extension for database files, since on Windows ME and Windows XP, any file that has this extension is automatically backed up by the System Restore facility.InterBase now recommends using ".ib" as the extension for database names.

  Our security database, formerly named *isc4.gdb* is now named *admin.ib*. For the present, the InterBase example databases may still have the ".gdb" name. In the future we will phase out that name and use new names.

- ODS11

  InterBase 7.0 introduces ODS11. This new On-Disk Structure is required by the presence of the new BOOLEAN data type and 68-byte meta-data names. To upgrade your databases, back them up with an ODS10 **gbak** and then restore them with the ODS11 **gbak** that comes with the InterBase 7.

- New name for the security database

  In InterBase 7, InterBase's security database is named *admin.ib* on all platforms. InterBase's internal tools have all been updated to use this name. If you wish to continue using your existing security database, you must back it up and restore it using the latest **gbak**. To change the name, specify *admin.ib* as the new name during the restore. If you have existing clients that expect to find *isc4.gdb*, you must update them to use the new name.

  You can specify a name of your choice for the security database by setting the ADMIN_DB parameter in the InterBase configuration file, *ibconfig*.

- New name for Unix configuration file

  On Linux and Solaris platforms, the InterBase configuration file was previously called *isc_config*. It is now called *ibconfig*.

- New keywords

  InterBase 7.0 adds the following new keywords:

  BOOLEAN TRUE FALSE UNKNOWN

  The following keywords were added to InterBase 6.5:

  ROWS TIES PERCENT

- New data type: BOOLEAN

  InterBase now supports a BOOLEAN audiotape, implemented to the SQL 99 standard.

  Examples:

```
CREATE TABLE AWARDS_1 (isEligible BOOLEAN, name VARCHAR(20));
INSERT INTO AWARDS_1 VALUES(TRUE, 'Jim Smith');
```

  INSERT INTO AWARDS_1 VALUES(FALSE, 'John Butler');

  SELECT * FROM AWARDS_1 WHERE isEligible = TRUE;

  ISQL and IBConsole return TRUE, FALSE, and UNKNOWN. Queries created with APIs return 1, 0, and NULL, respectively. For ESQL and DSQL programmers, we define the following type in *ibase.h*:

  define SQL_BOOLEAN 590

Note BOOLEAN is not supported in GPRE.

- No more SET TERM

  When you write SQL, there is no longer any need to use SET TERM to define a temporary terminator when defining stored procedures and triggers. InterBase now parses these statements correctly without the use of SET TERM.

  The document set states that IBConsole and IBX still require the use of SET TERM. InterBase believes that this was corrected after the document set was written and that SET TERM is no longer required in **isql**, IBConsole, or IBX. The old SET TERM functionality remains available in **isql**, IBConsole, and IBX, so that old scripts can still function.

- 68-byte meta-data names and XSQLDA

  Metadata names can now be 68 bytes long (67 bytes plus a null terminator). These names are available through all InterBase clients and are implemented in the new type 4 InterClient. They are being implemented in DBX and IBX and may be available by the time you read this.

The XSQLDA structure has been updated to support these long metadata names. Set the version field of this structure to SQLDA_CURRENT_VERSION to access long metadata names.

- New APIs for BLOBs and arrays

  Ten API calls that relate to blobs and arrays have been updated to support these longer metadata names. In these new APIs, the *desc* field points to an updated descriptor structure that accommodates long metadata names.

  The new API calls are:

isc_array_gen_sdl2()	*isc_array_get_slice2*()
isc_array_lookup_bounds2()	*isc_array_lookup_desc2*()
isc_array_set_desc2()	*isc_array_put_slice2*()
isc_blob_default_desc2()	*isc_blob_gen_bpb2*()
isc_blob_lookup_desc2()	*isc_blob_set_desc2*()

  The associated structure for arrays is ISC_ARRAY_DESC_V2. For blobs it is ISC_BLOB_DESC_V2. The associated defines are:

  #define BLB_DESC_VERSION2 2

  #define BLB_DESC_CURRENT_VERSION BLB_DESC_VERSION2

  #define ARR_DESC_VERSION2 2

  #define ARR_DESC_CURRENT_VERSION ARR_DESC_VERSION2

  These new API calls and their structs are documented in the *API Guide*. See Chapter 7, Chapter 8, and the new API calls in the API Reference chapter.

- Client version detection

  Some clients—notably drivers, but others as well—need to query the InterBase client library for the version numbers. Three new APIs provide this capability: *isc_get_client_version*(), *isc_get_client_major_version*(), and *isc_get_client_minor_version*(). They are described in detail in the API Function Reference chapter of the *API Guide*.

- New type 4 InterClient

  InterBase 7.0 introduces InterClient 3.0, This new version of InterClient is a type 4 JDBC driver, which means that it can communicate directly with the InterBase server. InterServer is no longer needed in environments where all the clients have been upgraded to this new type 4 InterClient.

To upgrade a client, place the new *interclient.jar* file on each client machine and ensure that it is the first instance on the CLASSPATH. Note that although the filename is the same as it was in earlier versions, the file being distributed with InterBase 7 is very different. It is a type 4 JDBC driver. Earlier versions were type 3. Be sure that you are installing the latest driver on your client machines.

Legacy InterServer: If you are not able to upgrade all of your clients immediately, install InterServer on the InterBase Server platform. InterServer will do no harm, and its presence allows a mixture of type 3 and type 4 clients to attach to the InterBase server. The *interserver.exe* file that distributes with InterBase 7.0 has not changed from previous versions.

- SMP support

  InterBase now provides symmetric multiprocessor (SMP) support for both clients and servers. Previous versions of InterBase ran on SMP systems safely by allowing only a single processor at a time to execute within the InterBase components. This release exploits SMP hardware by running InterBase threads on all processors simultaneously for increased throughput and performance.

  When you purchase a single server license, you acquire the right to use a single processor. You must purchase an additional license for each additional processor that you wish to use.

- Server configuration parameter: MAX_THREADS

  Setting the MAX_THREADS parameter in the *ibconfig* configuration file controls the maximum number of threads that can be active at one time within the InterBase engine. The default setting is 100:

  The ideal setting for this number depends partly on the nature of the work being performed by your clients. If you have many clients performing very similar tasks, you may want to lower the MAX_THREADS setting to reduce contention. On the other hand, if simultaneous activity is highly diverse, setting this to a higher value may increase throughput. This setting does not affect the maximum possible threads that can be created by the InterBase server but only the number that can be active in the engine at one time.

- Expanded processor control: CPU_AFFINITY

  On Windows multiprocessor platforms, you can specify which processors InterBase should use by adding the CPU_AFFINITY parameter to the *ibconfig* file. This setting is useful whenever the number of licensed processors is less than the number of actual processors present. When you purchase a single server license, you acquire the right to use a single processor. You must purchase one additional license for each additional processor that you wish to use.

  CPU_AFFINITY is discussed in the "Server Configuration" chapter of the *Operations Guide*.

- Increased security for external tables

Under some conditions, external tables could pose a security hazard. To counter this, InterBase has added the new requirements for external tables. External tables must meet one of the following conditions:

- The table is located in *<ib_home>/ext*. InterBase can always find external files that you place here.
- The location of the table is specified in the *ibconfig* configuration file by setting the EXTERNAL_FILE_DIRECTORY parameter to the location of the external file.

- New HTML reference docs

  InterBase now includes two references in HTML form. Both of these can be accessed from the Help menu of IBConsole or directly from the *<ib_home>/HtmlRef* directory.

  - **The SQL Reference** The *SqlRef.html* file replaces the older *SqlRef.hlp* file. It contains all the SQL statement information from the "SQL Statement and Function Reference" chapter of the *Language Reference*.
  - **The API Function Reference** The *APIFunctionRef.html* file is an HTML version of the "API Function Reference" chapter of the *API Guide*.

- Monitoring database attachments with system temporary tables

  The InterBase Server has always kept a lot of statistics about what was going on, but it has not been easy, or in some cases possible, to surface that information. InterBase now captures that information and makes it available in a set of global system temporary tables. These tables describe the runtime behavior of a database. They also provide a level of control. The temporary table metadata is listed in the *Language Reference*.

  It is also possible to exercise a certain amount of control over the state of a database by performing updates to these tables.

Table name	Description
TMP$ATTACHMENTS	One row for each connection to a database
TMP$DATABASE	One row for each database you are attached to
TMP$POOL_BLOCKS	One row for each block of memory in each pool
TMP$POOLS	One row for each current memory pool
TMP$PROCEDURES	One row for each procedure executed since the current connection began
TMP$RELATIONS	One row for each relation referenced since the current connection began
TMP$STATEMENTS	One row for each statement currently executing for any current connection
TMP$TRANSACTIONS	One row for each transaction that is active or in limbo

Querying system temporary tables Clients can query these tables using SELECT statements, just as they would query any other table. For frequent monitoring, the best transaction control is to start the transaction as READ_COMMITTED, READ_ONLY. Then commit it with COMMIT_RETAINING. This has the least impact on the system.

Updating system temporary tables By updating the TMP$STATE column of certain temporary tables, you can roll back an active or limbo transaction, commit a limbo transaction, cancel an attachment's executing operation, shut down the current attachment, or make an executing statement stop running.

System temporary table metadata The "System Tables" chapter of the *Language Reference* lists the metadata for each of the system temporary tables.

- Thread-safe processing of database handles

  InterBase 7.0 provides improved handling of InterBase database handles on behalf of client applications. Handle types include attachment, blob, BLR request, SQL statement, service and transaction objects. The client library manages the integrity of InterBase database handles in the face of concurrent application thread activity.

Index