INTERBASE

Versions 3.0 to 3.2 Release Notes

BORLAND

Disclaimer

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

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

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

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

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

Corporate Headquarters: Borland International Inc., 1800 Green Hills Road, P. O. Box 660001, Scotts Valley, CA 95067-0001, (408) 438-5300. Offices in: Australia, Denmark, France, Germany, Italy, Japan, New Zealand, Singapore, Sweden, Taiwan, and United Kingdom.

Software Version: V3.*n*

Current Printing: November 1992
Documentation Version: v3.3a1
Part Number: INT0033WW21690

How to contact Borland

Borland offers a variety of services to answer your questions about InterBase:

Customer support	Registered customers can call the InterBase customer support line at 1-800-437-7367 with their questions about InterBase. Also, general classes are available for InterBase training and consultants are available for individual customer site needs.
Marketing information	If you are not already an InterBase customer and want information about this product, call the product information line at 1-800-245-7376.
Documentation feedback	We welcome your feedback about errors or missing topics in the InterBase documentation.

Table of Contents

INTRODUCTION

This document includes all Version 3.*n* release notes and addenda distributed prior to the current release. Current release notes are in a separate document.

V3.2K Documentation Addendum

The following changes apply to InterBase V3.2K released on the Data General platform:

- InterBase V3.2K has been certified to run on the Sun Server 600 Series and the Sparc10 under OS 4.1.2-3.
- Journaling on VMS now works if enabled on databases larger than 64k.
- Bug number 3951 has been fixed so that memory is managed correctly when processing a trigger on a view on which additional processing was done prior to trigger activation.
- InterBase V3.2K now supports dynamic shared libraries which include dynamic function lookup on the Data General platform. Since this affects the manner in which user defined functions (UDFs) are handled, previously defined UDFs will not work in V3.2K and must be rebuilt. For an example of how UDFs are built and run in V3.2K, refer to *make.udf* located in */usr/interbase/examples*.

  In addition, linking your applications will require the following new options:

To use:	Link with:
shared libraries	-lgds
back end	-lgds_b -ldl
pipe server	-Bstatic -lgds -Bdynamic -ldl

To link your applications on Data General AViiON running DG-UX Version 5.4 or higher with the pipe server, use the following new options in the compiler command to use the pipe server rather than the shared library:

Language	Link with:
C	-Bstatic -lgds -Bdynamic -ldl
FORTRAN	/usr/interbase/lib/gds.a -ldl
C++	/usr/interbase/lib/gds.a -ldl

If you are linking your applications with **pyxis**, add the following options to theID command:

```
-lgds_pyxis -lcurses
```

- To enable **pyxis** on Data General, you will need to perform the following:
  - From within an **mterm** window, type the following command appropriate to the shell:

Command	Shell
setenv TERM vt100	cshell
TERM=vt100 export term	Bourne or kshell

  - Select the OPTIONS panel within the window and set the mode to vt100.
  - Use the keyboard mappings as displayed below or display them from the HELP menu:

vt100/vt52 key	AViiON key
PF1	ALT-Insert
PF2	ALT-PgUp
PF3	ALT-Delete
PF4	ALT-PgDn

A problem has been identified on the Data General platform when using nested UDFs. Building a shared UDF function library requires a workaround. If you have nested UDFs, you use both the **-Bsymbolic** and **-Bstatic** switches in the **ld** command in the order indicated in the example below:

```
ld -G -Bsymbolic udf.o -Bstatic -lm -lc -o <func_lib_name>
```

Your function library will be linked correctly; however, you will receive a warning message indicating, " _end is undefined in sbrk." This problem is currently being investigated by Data General. If you do not use nested UDFs, this workaround is not required.

V3.2J Documentation Addendum

The following release notes and bug fixes apply to V3.2J:

- A problem has been identified by HP with Apollo SR 10.4. Consequently, InterBase V3.2J on Apollo experiences problems running under this operating system and cannot be supported. HP will resolve this problem and provide users with a patch. InterBase customer support will notify you when the patch is made available by HP.
- The following bug is corrected in V3.2J:

  3855: Using the **sorted by** clause in a record selection expression with **erase**, then performing a **rollback** no longer results in corrupted data or a corrupt database.
- InterBase is available on 3.5 inch floppy diskettes on the IBM RS6000 platform. To install InterBase from floppy diskettes, use the following **restore** command instead of **tar**:

  ```
  restore -f /dev/rfd0h
  ```

  You then continue with the install procedure using the usual method.
- The following problem is known to exist on the SGI platform:

  Problem: User defined functions (UDFs) which make calls to routines in shared libraries compile and run correctly from user applications, but attempting to access the same UDFs in **qli** and **gbak** results in segmentation faults.

  Solution: If you encounter this problem on SGI, relink **qli** and **gbak** with your shared libraries.

  The command to relink **qli** is:

  ```
  cc /usr/interbase/lib/qlilib.a -lgds_s -lgdsf_s -lsun -o qli
  ```
- There is a 31-character restriction on module_name for user defined functions and filters. The module_name includes the path and the name of the module; for example, `module_name "/usr/gds/cd/kit_test/nfilter"`. If module_name exceeds the 31-character limit, the following error message appears:

  ```
  arithmetic exception, numeric overflow, or string truncation
  ```
- To run InterBase on the IBM RS/6000 platform, AIX 3.2 is the minimum version of the operating system supported.

Technical Bulletin

The following problem has been reported on the InterBase HM-V3.2J kit on the HP 9000/300, 400 platform running HP-UX version 8:

On HP-UX systems configured to use long filenames, InterBase returns the following message when attempting a remote connection:

```
connection rejected by remote interface
```

This problem results from the *inetd daemon* not being able to locate the TCP/IP server executable. The entry in */etc/inetd.conf* contains the path */usr/interbase/bin/gds_inet_server* when the actual path and filename on disk is */usr/interbase/bin/gds_inet_serve*.

To resolve this problem, perform one of the following tasks:

- Modify the entry in */etc/inetd.conf* from:

  gds_db stream tcp nowait root usr/interbase/bin/gds_inet_server gds_inet_server

to:

  gds_db stream tcp nowait root usr/interbase/bin/gds_inet_serve gds_inet_serve

- Or, move the executable to the long name by changing:

```
usr interbase bin gds_inet_serve
```

to:

```
usr interbase bin gds_inet_server
```

V3.2H Documentation Addendum

The following release notes and bug fixes apply to V3.2H:

V3.2H Bug Fixes

3368 **fred** properly handles large computed fields.

3369 (same description as 3368).

3388 **gpre** properly handles a missing logical operator (=) in SQL **where** clauses.

3401 Optimization corrected when accessing RMS-indexed files.

3435 Garbage collection for indexed nodes has been corrected.

3475 UDF definitions appear before field definitions in an extracted *.gdl* file.

3503 Records are not lost when a program does not include a **finish** statement.

3533 DSQL field names may be up to 31 characters in length.

3595 **gbak** converts floating point value -0 to 0.

3609 Missing dates are displayed as blanks.

3610 Invalid dates entered in a form are handled properly.

3618 **qli** correctly handles repeated calls to UDFs.

3628 **gpre** produces correct FORTRAN code on Sun.

3656 Lock table is correctly mapped on expansion.

3664 **gbak -r** no longer creates duplicate RDB$TRIGGER_MESSAGES.

3675 RDB$RUNTIME lists correctly on remote access.

3687 Optimization corrected when accessing RMS-indexed files.

3695 **gstat** **.gdb* no longer causes a VMS access violation.

3698 The last record stored in an external relation is written out on a **commit**.

3705 Optimization corrected when accessing RMS-indexed files.

3714 **show filters** properly displays filters.

3716 **grst** supports ODS-6 and ODS-7 databases.

3723 The example program, *dsql_date1.e*, correctly displays strings.

3727 The **gbak -r** option correctly restores databases containing generators equal to 0.

3732 The **gds_inet_server** detects partial data transfers and sends the remainder of the message.

3738 Dynamic array access works on Sparc platforms.

3743 **get_slice** and **put_slice** support **on_error**.

3744 *Interbase.Ada* supplies the correct data type for **get_slice** and **put_slice**.

3749 **put_slice** works correctly with remote connections.

3750 **gpre** generates correct code for **put_slice**.

3756 Triggers are executed in the order specified.

3757 **gpre** reports correct length for variables of type CHAR when processing C programs.

3761 Optimization corrected when accessing RMS-indexed files.

3764 "Record not found" is a legitimate status message when accessing RMS-indexed files.

3766 **gbak** correctly creates databases with reserves.

3769 **gpre** generates correct ADA code for both **compile_request** and **compile_request2**.

3770 **gpre** produces correct FORTRAN code on Sun.

3775 SCO UNIX platforms work correctly.

3778 Security across platforms correctly handles different group IDs.

3782 Embedded SQL query insertions into destination tables work correctly with 3475. UDF definitions appear before field definitions in an extracted *.gdl* file.

3785 **gbak** checks to see if RDB$RELATION_FIELD security classes are backed up.

Additional Fixes

- Reaching AST quota limits on VMS 5.4 now returns a bug check error. Users receiving this error should detach from InterBase to release their locks for other users.
- Potential loss of lock downgrade request signals no longer occur.
- **gdef** does not generate dollar signs ($) in DYN C++ code.
- The ACL filter prints out universal privileges.
- Index selectivity is used correctly by the optimizer.

VMS Support

InterBase V3.2H supports VMS 5.5.

Lock Table Expansion

Problems with lock table expansion that occurred in V3.2F on Sun and Apollo no longer occur in V3.2H. It is no longer necessary to force the lock manager to allocate a larger lock table.

Installation Changes

The installation procedure for SCO has changed. See *Installing and Running InterBase on UNIX* for these changes.

Using Forms on SGI and DG

If you are using **pyxis** in your program, the curses function leaves STDOUT in a nonbuffered mode when exiting a form. To return the terminal to line buffer mode, you must insert the following lines into your routine:

```
#include <stdio.h>
char buf[BUFSIZ];
setvbuf(stdout, buf, _IOLBF, BUFSIZ);
```

For example, the following routine is used to return a terminal to line buffer mode:

```
#include <stdio.h>
#include <curses.h>
main ()
{
char buf[BUFSIZ];
int ans;

initscr ();

clear ();

wrefresh (stdscr);

endwin ();

setvbuf(stdout, buf, _IOLBF, BUFSIZ);
printf ("Do you want to commit the updates (Y/N): ");
ans = getchar();
}
```

If you are using Forms with **qli**, you may get a blank screen after you accept or reject a form. To correct this, you exit **qli**. When you reenter Forms through **qli**, your updates will be visible.

Changes to *Interbase.Ada*

The following changes have been made to *Interbase.Ada* to support **gpre** preprocessing of ADA programs:

- If a blob is declared as text or the default subtype, for commands where **gpre** must declare a buffer for a blob, **gpre** now declares the type as a character vector. If the blob is declared as anything else, **gpre** now declares the type as a vector of byte integers.
- If a variable for fields is declared as type CHARACTER [n] SUB_TYPE FIXED, where n>1, **gpre** declares the variables as vectors of byte integers. Characters declared without SUB_TYPE FIXED will be declared by **gpre** as type STRING (1...n). Only SUB_TYPE FIXED gets changed.
- If a variable for fields is declared as type CHARACTER [1], **gpre** declares the variables as CHARACTER not STRING (1...1).
- If a variable for fields is declared as type CHARACTER [1] SUB_TYPE FIXED, **gpre** declares the variable as a byte integer.
- The following parameters are now declared as ISC_USHORT (unsigned short) rather than SHORT_INTEGER, since they must be picked up as unsigned short by InterBase functions:

```
ATTACH_DATABASE
   file_length
   dpb_length
COMPILE_REQUEST
   blr_length
COMPILE_REQUEST2
   blr_length
CREATE_BLOB2
   bpb_length
CREATE_DATABASE
   file_length
   dpb_length
   dpb_type
DDL
   msg_length
```

```
EVENT_BLOCK (RETURN value)
   count
EVENT_COUNT
   count
EVENT_WAIT
   length
GET_SEGMENT
   buffer_length
   actual_length
GET_SLICE
   sdl_length
   param_length
OPEN_BLOB2
   bpb_length
PREPARE_TRANSACTION2
   msg_length
PUT_SEGMENT
   length
PUT_SLICE
   sdl_length
   param_length
QUEUE_EVENTS
   length
RECEIVE
   msg_type
   msg_length
SEND
   msg_type
   msg_length
START_AND_SEND
   msg_type
   msg_length
START_MULTIPLE
   count
START_TRANSACTION
   tpb_length
BLOB_INFO
   msg_length
   buffer_length
REQUEST_INFO
   instantiation
```

```
    msg_length
    buffer_length
DATABASE_INFO
    item_length
    buffer_length
DSQL_EXECUTE_IMMEDIATE
    command_length
DSQL_PREPARE
    command_length
COMPILE_MAP
    map_length
COMPILE_SUB_MAP
    map_length
CREATE_WINDOW
    name_length
    width
    height
MENU
    menu_length
DRIVE_MENU
    blr_length
    title_length
    terminator
    entree_length
GET_ENTREE
    entree_length
    entree_end
PUT_ENTREE
    entree_length
```

- Parameters to InterBase procedures and functions are now declared as generic data types (ISC_SHORT, ISC_LONG) appropriate to ADA Implementor-specific data types (SHORT_INTEGER, INTEGER).

- Since **gpre** processed calls to low-level entry points as if they were high-level commands, the prefix (isc) has been added to the calls to differentiate them from the high-level commands:

```
ISC_CANCEL_BLOB
ISC_CLOSE_BLOB
ISC_COMMIT_TRANSACTION
ISC_CREATE_WINDOW
ISC_DELETE_WINDOW
ISC_EVENT_WAIT
ISC_GET_SEGMENT*
ISC_GET_SLICE
ISC_OPEN_BLOB
ISC_PREPARE_TRANSACTION
ISC_PUT_SEGMENT*
ISC_PUT_SLICE
```

  * The type declared for buffer parameters for **isc_get_segment** and **isc_put_segment** is SYSTEM.ADDRESS, so that a buffer may be of any data type required.

V3.2F Documentation Addendum

The following release notes and bug fixes apply to V3.2F:

- C++ support is available as a separately-priced InterBase item. Please contact your sales representative for pricing and platform availability.
- Open Windows support is now provided on Sun. Refer to the *Installing and Running InterBase on Sun* document for information.
- In a **qli** query involving an aggregate (min, max, total, etc.), the default column header is now the name of the aggregate.
- UDFs now have a ten argument limit.
- On HP-UX systems, remote events are unreliable. Local events are not affected. This has been discovered to be an HP-UX operating system problem. We are currently consulting with HP to determine whether or not the kernel can be reconfigured to eliminate the problem.
- Remote events now work on the DG-UX V3.2F kit.
- On DG-AViiON, the C++ SQL examples do not work because of an AT&T cfront v2.1.1 compiler problem. The problem will be corrected in cfront v2.1.2.
- On Motorola Delta and IMP kits, **pyxis** is not supported.
- The total key length permitted for a compound index has been reduced to 202 characters in V3.2F. This limitation is a result of bug #3645. **gdef** will now reject definitions for compound indexes longer than 202 characters. Indexes defined prior to V3.2F may have values that exceed the new maximum. For these indexes, attempts to rebuild the index or backup and restore the database will fail with the following error message:

  ```
  gds_$keytoobig error,
  -key size exceeds implementation restriction for index
  <index_name>
  ```

  If this error occurs, you should shorten the length of the fields involved in the index to ensure that they meet the new limit or drop any unnecessary fields from the compound index.
- The name of the lock table on the Sun platform has been changed from:

  /usr/interbase/gds.lockfile.<*node_name*>

  to

  /usr/interbase/gds.lock.<*node_name*>

  This brings the Sun platform into line with InterBase naming conventions.

The change should be transparent to all applications that link with either the **-lgds** or **-lgdslib** linker options. However, any applications linked with the *gds_b.a* library, via the **-lgds_b** linker option, **must be relinked before they will work with V3.2F**. Attempts to use applications that have not been relinked will fail with a "no permissions" error when they attempt to access the lock table.

- **All Sun and Apollo users should check the size of the lock table.** In V3.2F, to avoid problems with lock table expansion, most notably, failure of the **gds_lock_print -a** command, InterBase provides a method to force the lock manager to allocate a larger lock table.

  All Sun users should check the current size of:

  /usr/interbase/gds.lockfile <*node_name*>

  Apollo users should check:

  /sys/node_data/gds.lockfile4

  If either of the files is larger than 32,768 bytes, create a larger lock table to avoid problems with lock table expansion. To do so on Sun:

  — Copy the *lock_header_template* in */usr/interbase* to */usr/interbase/lock_header*

  — Increase the SHMSIZE amount by 32,768 bytes or 32K. It is recommended that additional increases be in multiples of 32K.

  To create a larger lock table on Apollo:

  — Copy the *lock_header_template* in */interbase* to */interbase/lock_header*

  — Increase the SHMSIZE amount by 32,768 bytes or 32K. It is recommended that additional increases be in multiples of 32K.

- On all Sun platform installations, InterBase, by default, is installed on */home/*<*server_name*>. **You cannot override the default by installing InterBase in */usr* without causing problems with the installation**. However, if you choose to install InterBase on the */usr* partition, you must create the directory */usr/IB* before you run the installation process and then enter that directory name during installation when asked:

```
This installation will create the directory tree
<directory _tree> under the root directory most
appropriate for your installation.Enter the root
directory you wish to use [home/server_name>]:
```

- In V3.2F, to link your application on DG-AViiON running DG-UX Version 5.4 or higher with the shared access method library, add the following options to the **ld** command:

```
-lgdslib -lgdsflib
```

  If you are linking your application with **pyxis**, add the following options to the **ld** command:

```
-lgdslib -lgdsflib -lgds_pyxis
```

- On the Apollo DN10000 AP_3.2F kit, you may not get the correct DSQL results with FORTRAN if you are using the 10.5 version of the FORTRAN compiler. To avoid this problem, you should run the compiler using the **-dba** option.

V3.2F Bug Fixes

648	**modify trigger** on views with computed fields no longer gets a MOV_move conversion error.
735	See bug 648.
921	**gds_$decode_date** now correctly stores the day of the week.
2403	See bugs 735 and 648.
3404	SQL **select** containing "running count" now works correctly in **qli**.
3414	InterBase no longer gets an access violation when updating a view with a computed field.
3418	Storing and modifying a record in the same transaction, so it violates a unique index, no longer causes the record to disappear.
3434	**qli** now displays array index information.
3438	**select** on a unique descending index now displays all records.
3496	InterBase now uses substantially fewer locks for events under VAX/VMS.
3505	See bug 3438.
3514	Defining a relation in **qli** using **based_on** no longer results in a segmentation violation from buffer overflow.
3518	**gfix** handling of limbo transactions has substantially improved.
3521	"**based_on** *<field_reference>*" in a **qli** procedure now works in nested procedures and **begin-end** blocks.
3524	**qli** now allows using array fields as arguments to UDFs.
3531	The RDB$_DESCRIPTION field is now properly stored as a subtype 1 (text).
3535	See bug 921.

3539	**gdef** now extracts the correct definition of a UDF with blob arguments.
3548	See bug 3531.
3553	A segmentation fault after finishing one of two databases no longer occurs.
3555	A large volume of stores via **qli** script no longer causes the **gds_inet_server** to fail.
3556	On a DEC Ultrix machine with a blob filter of subtype 0 to subtype text now works correctly.
3558	See 3555
3562	Events now work correctly on multi-hop connection (e.g., **gds_inet_server** to **gds_server**).
3566	Asynchronous events on the VAX now work correctly and consistently.
3567	Event programs that post events before listening for them now work correctly.
3574	Multi-RDB database programs now work correctly.
3575	The **qli define relation**, based on a non-existent relation, now returns an error.
3577	The **nested for** construct is now handled correctly by **qli**.
3581	Events are handled properly when a remote database is finished and rereadied.
3583	**gbak** and **gdef** now create databases with default protection of 666.
3584	Memory usage growth no longer occurs when using UDFs.
3585	Nulls are no longer allowed in a unique index.
3590	In **pyxis**, access rights are now maintained when one user switches (su) to another user.
3596	**gbak -restore** now works correctly with generators and no longer runs out of memory.
3597	**gconv** now returns an error if missing "end of blob marker" from data file.
3599	**gconv** now handles blobs correctly on non-Apollo machines.
3624	**gpre** now generates legal initialization of BLR values for Apollo FORTRAN compilers.
3609	Missing value DATE (i.e. 17-Nov-1858) is now displayed correctly.
3611	See bug 3592.
3610	Invalid date no longer causes forms to hang.
3617	An RSE that queries for a date_fld = "01/01/00" now selects the correct records.
3626	Remote security now works correctly.
3630	Read/write now guaranteed to complete with SunOS.

3633 Event counts are now incrementing correctly when multiple events are posted within the same transaction.

3637 VMS navigational equality queries now return the correct records.

3640 **gdef -dyn** now supports C++ code generation via a **-cxx** switch.

3641 **gpre** no longer converts dollar signs ($) to spaces in C++ code.

3643 ***gds.hxx*** for C++ now has the correct prototype for **isc_create_database**.

3645 Compound index keys no longer cause "keytoobig errors" or segmentation faults for valid indexes.

3647 **gbak** now supports more than 1032 fields in a single relation when backing up databases.

3651 **qli** no longer truncates exponents when printing double float quantities.

3658 Conversion errors no longer permit duplicate records to be stored in unique indexes.

3678 Array references in C++ RSEs are now 0 based, not 1 based.

3697 On DGUX, remote NFS databases are now being accessed correctly.

3699 See bug 648.

3709 Field-level security now works.

VERSION 3.2 RELEASE NOTES

Overview

This document describes changes that have occurred since the InterBase Version 3.1 release:

- New features for InterBase Version 3.2, including new language support
- Software restrictions and suggested workarounds, when available
- Bug fixes

Documentation corrections and clarifications are in the *V3.2 Documentation Corrections* document.

Version 3.2 Features

New Language Support

C++

Saber C++, Version 1.02 is available on the Sun/3 and Sun/4 and on Apollo HP-UX.

ADA

Verdix ADA, Version 3.0 is available on the Sun/3. (Telesoft ADA is no longer supported.

gpre Additions

Specifying Cache Size

For programs that use many databases but access or change only one or a few relations in some of the databases, a smaller cache size can be used for those databases. For programs that access or change many records in many databases, performance may be better with a cache size greater than the default for those databases.

You can specify the database cache size count (i.e., the total number of buffers) when opening existing databases with the **gpre ready** statement or the OSRI **isc_attach_database** call. The syntax is:

```
READY database-specs [DEFAULT CACHE n [BUFFERS]]
    [on-error clause]

database-specs::
    database-id [CACHE n [BUFFERS]]
```

where *n* is an integer. The minimum number of buffers (*n*) is 10 and the maximum depends on the maximum allowed on your particular system. If you specify a cache for a database, that cache size applies only to that database. If you specify a default cache, that cache size applies to all databases listed in the **ready** without their own specific caches. If you do not specify a cache count for a particular database, the default cache count of 75 is used.

For example,

```
READY DB1.GDB CACHE 80, DB2.GDB
```
opens DB1 with 80 buffers and opens DB2 with the default of 75 buffers

```
READY DB1.GDB, DB2.GDB CACHE 50 BUFFERS, DB3.GDB DEFAULT CACHE 80
```
opens DB1 and DB3 with the new default of 80 buffers and opens DB2 with 50 buffers.

```
READY DEFAULT CACHE 50 BUFFERS
```
opens all known databases with the new default of 50 buffers.

Each single-user attachment maintains the specified cache size as long as its attachment is active. Attachments through a server maintain the specified cache size as long as there are any server attachments to the database. If a database is already attached through a multi-client server, an increase in cache size which results from a new attachment will persist until all the attachments end. A decrease in cache size will not affect databases that are already attached through a server.

Specifying Databases with start_transaction using Clause

You can list databases with the new **using** clause in the **start_transaction** statement. The database handles listed in the **using** clause identify only the databases to be affected by this transaction. **Using** allows a multi-database program to start transactions against a subset of its databases. In contrast to the **reserving** clause, which restricts the database relations that the transaction can access or modify, the **using** clause limits the databases involved in a transaction. The **using** and **reserving** clauses are mutually exclusive. The syntax of **start_transaction**, with the new **using** clause is:

```
START_TRANSACTION [transaction-handle]
    [CONCURRENCY | CONSISTENCY] |
    [WAIT | NOWAIT] |
    [READ_WRITE | READ_ONLY] |
    [USING dbhandle-commalist | reserving-clause]
    [on-error clause]
```

General information about **start_transaction** begins on page 4-102 of the *Programmer's Reference*.

Increased ANSI FORTRAN Compatibility

gpre will correctly handle the '!' delimiter for inline comments in FORTRAN programs on all platforms (i.e., VMS, Apollo, SUN/3, SUN/4, SUN/Sparc, HP, and SGI).

gbak Addition: blocking factor

A new **gbak** switch, **-factor**, has been added to allow you to control the blocking factor when you back up a database to tape. This switch, which can be abbreviated as **-fa**, increases the speed of the backup and allows more data to fit on the tape. You do not need to specify the **-factor** switch when restoring a database; InterBase automatically interprets the blocked data.

gdef Addition: set generator

The default initial value of a generator is 0. The following **gdef** syntax allows you to set the starting value of a generator. The starting value can be zero, a negative integer, or a positive integer. (The maximum and minimum values are equal to the limit on the size of a generator.)

```
SET GENERATOR <generator name> [TO] <integer>
```

The next value generated will equal the new value plus one.

This syntax can only be used in **gdef** and you must have write privilege for the system relation, RDB$GENERATORS; you cannot set the starting value of a generator in **qli**.

gdef Change: UDF parameters

In Version 3.2 up to ten parameters can be used with UDFs; **gdef** will return an error if you attempt to define more than ten parameters.

NOTE

If the UDF returns a blob, only 9 parameters are allowed.

The following rules apply to UDF parameter passing:

- All calling parameters are passed by reference.
- Numerics (short, long, float, and double) can be returned by value or by reference. Shorts and longs are returned as longs, and floats and doubles are returned as doubles.
- UDFs cannot return arrays.
- Arrays and blobs are passed as references to array/blob UDF structures.

SQL Error Reporting Change

A new function, **isc_print_sqlerr** prints the SQLCODE, an SQL error message, and the InterBase error message if the status vector indicates that there is a relevant message. The SQL messages are stored in the message database under their SQLCODE numbers. The syntax of the **isc_print_sqlerr** is:

```
isc_print_sqlerr(sqlcode, status_vector)
    short   sqlcode;
    int     *status_vector;
```

This function was added because setting an SQLCODE in embedded SQL applications does not always set a corresponding status vector value, which is used by **isc_print_status** to display a message.

Another new function, **isc_sql_interprete** has been added to Version 3.2. This function retrieves an SQL error message into a user-supplied buffer. (A buffer length of 128 should be sufficient to hold the message.) This will allow programmers to build error display routines. This syntax of **isc_sql_interprete** is:

```
isc_sql_interprete (sqlcode, buffer, length)
    short   sqlcode;
    text    *buffer;
    short   buffer_length;
```

In addition, when the status vector element of a **isc_print_status** is 0 (indicating that no failure occurred), instead of displaying the message, "message text not found", the message, "Success" will be displayed.

SQL Security Changes

Field-level **update** privilege has been added to the **grant** statement of DSQL. This allows restricting users to updating or referencing only certain fields in a relation.

insert and **update** privileges for a relation no longer require **select** access. This change also applies to access control lists. When a field in a relation is referenced, InterBase checks to see if it is part of a **store** or a **modify**. If it is, **insert** or **update** privilege is required for both the field and its relation; otherwise, **select** is required for the field and its relation.

DSQL Addition: column aliases

You can enter a column alias token immediately following the selected field or expression in dynamic SQL queries. This column alias token will be returned as a string in the SQL name field of the SQL var. The column alias cannot have commas or quotation marks surrounding it, nor can it have any blanks in it. If an expression is used, no column name is returned, and if a field is used, the name of the field is returned. For example:

```
select num, 2.2*weight weight_kg from p
```

returns a list of weights defined by the expression "2.2*weight" with each selected expression having the column alias "weight_kg".

Dynamic Access to Arrays

Arrays in Version 3.1 could be accessed from 3GLs either automatically (with a simple array reference) or manually (with **get_slice/put_slice**), neither of which is appropriate for dynamic access. The following information describes a properly-layered library of functions to provide runtime access to arrays. (There is no preprocessor support for these calls.)

The central structure in the library is an array descriptor, which is used to describe an array or an array slice. The descriptor can be initialized by one of four methods:

- By the function **isc_array_lookup_desc** -- this function does a metadata access to get the data type, length, scale, and dimension for the named field and relation. The *field_name* and *relation_name* may be either null-terminated or blank-terminated.
- By the function **isc_array_lookup_bounds** -- this function is similar to **isc_array_lookup_desc**, but also fetches the array bounds as defined in RDB$FIELD_DIMENSIONS. The bounds information is not required for other functions in the library, but may be useful to the application.
- By the function **isc_array_set_desc** -- this function initializes the descriptor from the function parameters, without reference to the database metadata. The data type is given as a SQL data type number (i.e., SQL_TEXT is 452). Null indicator values (SQL_TEXT +1) are ignored and treated as 452.
- Directly by the program -- the *array_desc_dtype* is expressed as a BLR data type and *array_desc_field_name* and *array_desc_relation_name* must be null terminated.

The following fields and functions are used in the structure shown below:

The structure field, *ARRAY_DESC_FLAGS* controls whether the slice is fetched in column major or row major order. The three descriptor initialing routines set the field to the default setting (appropriate to the C language). If the array is to be fetched for FORTRAN, the field is changed by the host program to the value 1.

The function **isc_array_gen_sdl** can be used to generate slice description language (SDL) for use with the actual **isc_get_slice** or **isc_put_slice** calls. The function is most useful for avoiding SDL generation inside a loop.

The function **isc_array_get_slice** is used to fetch an array slice. The bounds information in the array descriptor defines the slice to be fetched. Note: The function interprets the slice bounds in the same dimension space as the original array. In **gpre** array support, bounds are automatically adjusted to language specific ranges (in C, all arrays have zero as a lower bound; in FORTRAN, all arrays have one as a lower bound). In the function **isc_array_get_slice**, the parameter SLICE_LENGTH is used to

specify the size of the resultant slice (in bytes), and is returned as the number of significant bytes fetched.

The function **isc_array_put_slice** is analogous to **isc_array_get_slice**, but runs in the other direction. **isc_array_put_slice** also returns an array ID.

```
typedef struct {
   short          array_bound_lower;
   short          array_bound_upper;
} ARRAY_BOUND;

typedef struct {
   unsigned char  array_desc_dtype;
   signed char    array_desc_scale;
   unsigned short array_desc_length;
   char           array_desc_field_name [32];
   char           array_desc_relation_name [32];
   short          array_desc_dimensions;
   short          array_desc_flags;
   ARRAY_BOUND    array_desc_bounds [16];
} ARRAY_DESC;

#define ARRAY_DESC_COLUMN_MAJOR    1   /* Set for FORTRAN *

isc_array_lookup_desc (status, db_handle, trans_handle,
   relation_name, field_name, desc);
   long           *status;
   long           *db_handle;
   long           *trans_handle;
   char           *relation_name;
   char           *field_name;
   ARRAY_DESC     *desc;

isc_array_lookup_bounds (status, db_handle, trans_handle,
   relation_name, field_name, desc);
   long           *status;
   long           *db_handle;
   long           *trans_handle;
   char           *relation_name;
   char           *field_name;
   ARRAY_DESC     *desc;
```

```
isc_array_set_desc (status, relation_name, field_name,
   sql_dtype, sql_length, dimensions, desc);
   long            *status;
   char            *relation_name;
   char            *field_name;
   short           *sql_dtype;
   short           *sql_length;
   short           *dimensions;
   ARRAY_DESC      *desc;

isc_array_gen_sdl (status, desc, sdl_buffer_length,
   sdl_buffer, sdl_length);
   long            *status;
   ARRAY_DESC      *desc;
   short           *sdl_buffer_length;
   char            *sdl_buffer;
   short           *sdl_length;

isc_array_get_slice (status, db_handle, trans_handle,
   array_id, desc, array, slice_length);
   long            *status;
   long            *db_handle;
   long            *trans_handle;
   GDS_QUAD        *array_id;
   ARRAY_DESC      *desc;
   void            *array;
   long            *slice_length;

isc_array_put_slice (status, db_handle, trans_handle,
   array_id, desc, array, slice_length);
   long            *status;
   long            *db_handle;
   long            *trans_handle;
   GDS_QUAD        *array_id;
   ARRAY_DESC      *desc;
   void            *array;
   long            *slice_length;
```

Limbo Transaction Processing Change

In Version 3.1, when a multi-client server lost a connection (i.e., detected a communication error) to a client performing a two-phase commit, it terminated the database attachment without releasing any limbo transaction locks. This caused other users to become suspended if they tried to modify records updated by the limbo transaction. (Usually, a **gds_$deadlock** error code was returned with a minor code of **gds_$trainlim**.)

In Version 3.2, **isc_detach_database** releases the locks on limbo transactions. Releasing the lock still leaves the transaction in a state of "limbo"; the normal procedures and tools for handling limbo transactions must be used. Refer to the *Database Operations* manual, pages 3-8 for information on limbo transactions.

Floating Point Numbers

In Version 3.2, floating point numbers are more accurate. Although accuracy is improved, equality comparisons between floating points are not recommended due to the nature of floating points. For example, instead of this comparison:

```
if variableA = 123456.123445
```

a comparison that includes an error factor should be used:

```
if (variableA - 123456.123445 < 0.000005)
```

Technical Bulletin Update

The architectural limitation on the number of transactions noted in the Technical Bulletin dated April 11, 1991 has been lifted. While you should continue to periodically sweep your database, failure to do so will no longer risk losing data due to an excessive number of transactions.

V3.2 Bug Fixes

The following bugs have been fixed in Version 3.2. Bug fixes that improve NIST SQL compliance are marked.

Bug #	NIST	Bug Fix Description
224		Pressing ^Z within the CON> prompt no longer causes **qli** to exit.
363		Union is now supported in DSQL.
543		Appropriate message is now returned when deleting records from an external file within **qli**.
563		Entering a query_header clause in a field definition in **qli** now returns the proper error message.
685		A syntax error in defining an index no longer results in an incorrect index definition.
691		A form will only be used in a **modify** if a form is explicitly named or when forms are set and no fields are listed in the **modify**.
703		Set Form and **print... on** <*file_name*> now directs the output to the designated file.
706		Set Form and **list... on** <*file_name*> now directs the output to the designated file.
740		After an internal **gds** consistency error, the correct error message is now displayed in **qli**: "can't continue after bugcheck".
756		When accessing a relation by **db_key** in C, **gpre** now assumes that the host language variable which currently contains or is about to receive a **db_key** is long enough to hold the key.
859		**gpre** and FORTRAN now handle date copying correctly.
897		View deletion performance has been improved.
904		**qli** now recognizes Sun pathnames starting with '~'.
933		Conversion errors no longer occur when using floats with edit strings containing "$".
937		Data typing of DSQL expressions has been improved (expressions will not automatically become varying [80]).
1017	x	Testing the indicator variable after set function on an empty table no longer gives an incorrect value. It is now <0.
1503	x	A subquery that selects no records no longer returns an error message, but evaluates the comparison operator to unknown.
1540	x	**count** on an empty table now returns 0, instead of no value.
1545		Using **fred** now requires **qli**, data definition, or **gpre** to be licensed.

1558	FORTRAN **implicit none** now declares and uses variable GDS_$I as the index variable in data statements.
1598	**include sqlca** and **database** statements can both be included in a program.
1608	**gpre** now detects a semi-colon at the end of a commit, rollback, save, or prepare statement within a host language **if** statement.
1626	A security class not allowing modifications to data definition now does not allow any changes to global fields either.
1700	Edit strings requiring more precision than available by the data type/value no longer print as **, but print with the amount of precision available.
2540	Zero-length item descriptions (in brackets) are correctly retained by **gdef**.
2881	Subqueries that select expressions of aggregates work correctly in **gpre**.
2897	**store using** <*blob field*> = **edit** in **qli** works correctly.
2900	Context variable with same name as blob name works correctly in **gpre.**
2931	Failed database attaches now report the correct database name.
2946	For tasks that **gdef** cannot complete (due to missing info), **gdef** will ignore/clean up any starting steps it took.
2959	Views that contain a union and computed expressions are now retrieved correctly by DSQL.
2968	Maximum fill string length has been expanded to 255 characters (from ~30).
2994	The **gpre** Pascal preprocessor correctly handles multiple databases in the **gds_$start_transaction** argument list.
3007	**gdef** now handles interrelationships so interdependent objects can be manipulated (e.g., **delete trigger** and then delete its relation).
3026	**qli** commands longer than 255 characters now produce an error message.
3030	Support for old "*gds.authorize*" licenses is no longer available in V3.2.
3032	Aborting from an editor when storing a blob now marks the blob as missing.
3035	Defining a relation or view and then deleting it within the same execution of **gdef** is no longer permitted.
3042	**gpre** now generates correct code when a host variable specified as a target of SQL **fetch** is an array.

3044	Asking for a very large number of buffers at database attachment now works correctly.
3051	**gdef -e** now extracts views from V3.0 databases in the correct order.
3072	Report writer now correctly formats output when a query contains a three-way join where one of the joins is a self join.
3113	Remote events now work correctly.
3121	**gpre** now processes Pascal comments correctly.
3130	An error message is now displayed when the user attempts to define multiple triggers in a single **define trigger** statement in **gdef**.
3148	In **qli**, a **repeat store** using a form with a blob now works correctly.
3178	In a program which does only dynamic SQL, when a blank database is found, no automatic **attach database** will be generated.
3194	In **gdef,** deleting a relation with a computed field now works correctly.
3198	**edit** command now works correctly when remotely logging on to an Apollo.
3200	The output produced by a **show relations** executed while logged onto an Apollo using a VT100 or emulator is now correctly displayed.
3220	**gpre** now parses **distinct** in a subquery.
3223	The DECnet pipe server is no longer killed when user finishes db1 and then readies db2 ("broken pipe message" will not appear).
3230	In large databases (> 200M) on Apollos, queries will no longer fail with "reference to illegal address" message.
3239	RDB$USER_NAME is now correctly maintained when going from a TCP to an MBX server.
3255	Using a syntactically incorrect **create index** command in the *full_dsql* example now returns a syntax error, rather than crashing.
3256	In **gpre**, lengths of CHAR and VARCHAR in **declare table** have been adjusted so syntax errors do not occur.
3257	In **gpre**, declaring a table that already exists now returns an error message rather than crashing.
3259	**qli** now correctly shows the database handle in the **show** <*db_handle*>.**indices** command.
3260	**iscinstall** now accepts non-default Ethernet ID.
3261	The DSQL **between** clause now works correctly.
3262	Low-level DSQL calls now work correctly with multiple databases that have same object names, but different object definitions.
3263	Character-special ("raw") UNIX I/O interface is now supported.

3264	Remote TCP server now works correctly with events and does not segmentation fault when running remote events.
3265	Modifying a database with DYN now correctly defines triggers if the DYN also contains a definition of the generator to be used by the trigger.
3267	More than 100 events now work correctly on the Apollo and do not produce an "Apollo-specific fault".
3275	*<Field>* EQ null or *<field>* NE null now works in **gdef** validation clauses.
3276	Pre-erase triggers on views now work correctly.
3277	On VMS, **gbak** now creates versions of a **gbak** file, instead of writing over existing backup.
3279	Logging now works correctly if attaching more than one database in **qli**.
3280	Date edit strings in X(n) format now work correctly, instead of occasionally causing **qli** to crash.
3282	**qli** edit now brings up the correct number of commands in a window.
3284	**gbak -t** no longer nulls out restored array data.
3286	SQL union with **group by** now works correctly.
3290	DSQL now handles multiple metadata changes correctly.
3291	Aliases are now allowed in a **group by** clause in **qli**.
3292	Licensing check stop message has been changed to report registration hours as 9-5.
3293	Installation procedure now correctly defines GDSSHR_TCP before running **gbak**.
3295	Unknown remote hosts on GDSSHR_TCP now work correctly.
3299	Discarded pointer page referenced by index is now ignored.
3301	DSQL TIME can now be cast as TEXT for input (but cannot use blanks to separate date and time; must use slash, hyphen, comma, or colon).
3303	**gfix -h** interval defined in GDS_$INFO_SWEEP_INTERVAL now survives **gbak**.
3308	**gbak** now handles multidimensional arrays correctly.
3309	**dsql_$finish** now works correctly and does not cause memory growth of process.
3310	**qli** unknown switches are ignored and will no longer cause **qli** to terminate.

3313	A quoted string entered as a query name now returns an error message.
3314	Missing value flags declared as text strings of length 1 now work correctly and use "1" to indicate the missing value.
3315	**edit** command in **qli** now works correctly with EMACS.
3316	**gdef -e** now extracts all fields with system flags not equal to 1, instead of only 0 and -1.
3318	**qli spawn** now runs in parallel, not supervisor (blocking) mode.
3319	Loading records from external relations via a procedure now works correctly.
3321	*Stocks.gdl* in examples has been corrected and now matches restored *stocks.gbak*.
3322	**gstat** version number switch **-z** has been added.
3326	**gdef** prints an error message for illegal syntax in trigger messages.
3330	Generated DYN specifies index type (ascending/descending) when needed and now creates indexes correctly.
3337	**gpre** now correctly processes FORTRAN boolean operators.
3338	**qli** now correctly processes UDFs that pass last parameter as NULL.
3339	DSQL now returns an error message if you create a relation and attempt to insert data before committing (no longer produces "can't find pointer page" message).
3340	**gdef -e** now correctly extracts SQL security on SQL defined tables.
3341	DSQL now correctly assigns SQL security to new tables.
3343	Entry points for **isc_encode_date** and **isc_print_blr** have been added to VMS *gdsshr.exe*.
3347	**qli print** literal limit has been expanded to 255 characters and an error message is displayed if more than 255 characters are used.
3348	**qli** no longer drops chars when using print concatenation and UDFs.
3349	On IBM RS/6000 **gconf trim()** function now correctly prints the string it returns.
3351	**qli** field position number is now assigned explicitly only when relation is defined; if not specified, position number is "missing".
3352	In **qli**, **show variable** is now supported as well as **show variables**.
3353	In **qli**, the parser now accepts only one data type definition in variable declarations; an error message is displayed if more than one data type is entered.
3354	VMS TCP connect now checks **getservport errno** when establishing outgoing TCP connection, producing a more specific error message.

3355	Event names are no longer truncated at the first space encountered.
3357	Undocumented upper limit in **isc_event_block**() has been removed, allowing users to wait on more than 64 events.
3358	**qli** now correctly sorts data when using UDFs.
3359	DYN support has been added to **modify database** command.
3361	**gpre** now correctly handles host variable references on the left side of list queries.
3364	**qli print** command now correctly prints blank lines in blobs.
3370	Individual slices of arrays with varying strings are now aligned correctly on SPARCs.
3373	**gdef -dynamic** now supports descriptions on **define** and **modify trigger** statements.
3376	**qli** now correctly compares date variables and date strings in sub-queries.
3378	**gltj flush** in batch mode on VMS now works correctly.
3379	**gpre** now handles multiple database statements in FORTRAN correctly.
3389	Moving past the bottom of a form is now permitted in **fred**.
3391	Blob handles are now correctly generated for ADA (no longer generating **gds_handle** for blobs).
3394	Error handling has been added to **commit** and **rollback** for SQL.
3396	(same as 3280)
3397	(same as 3339)
3398	Querying newly-created tables in DSQL before **commit** now returns an error message.
3402	*InterBase.a* and *InterBase.ada* now correctly call **dsql_prepare**.
3406	**gpre** now allows SQL indicator variables to be any valid host language data type (including array elements).
3410	Users are now permitted to manipulate data in a view if they have read/write permission to the view, regardless of whether they have access to the relation.
3412	Corrected spelling error in DSQL_DECL_ERR message.
3413	**Commit**s in **qli begin-end** blocks now work correctly (no longer cause "invalid transaction handles" error).
3430	Relation names with spaces now work correctly.
3431	**isc_set_debug** has been added to V3.2 kits.

3432	**get_slice** now works correctly with range bounds specified as host language variables.
3444	Port blocks that contain non-text data now work correctly on ADA.
3445	Files are now extended by 64KB beyond the page being written prevents database corruption from allocated but not formatted pages).
3450	**gpre** C++ now correctly parses printf statements with \ in them.
3451	(same as Bug 3223)
3453	Post-store triggers on views now work correctly.
3454	In *dsql.ef* example, **baddress** calls have been changed to **isc_baddress**.
3456	Expressions of aggregates now work correctly in DSQL.
3458	**gpre** now correctly processes FORTRAN programs that use the FORTRAN keyword **save,** but the **transaction_handle** keyword must be used to save a named transaction.
3460	*gds.ins.ftn* FORTRAN include file now compiles with **-ff** switch to **ftn**.
3466	Aborted **gdef** database creation now deletes secondary files of the database and, if created, shadows.
3469	**gbak -r -k** no longer restores any shadow metadata to the new database.
3470	Corrected the syntax error in help text for the **insert** statement.
3477	In DSQL, expressions of parameters now work correctly.
3478	Inputting a long in SQLDA to a short in the database or outputting a long in the database to a short in SQLDA is now reported as a conversion error.
3481	Ultrix automounts now work correctly.
3483	(see Bug 3339)
3496	Events on VMS no longer require excessive locks.
3501	Incorrectly placed commas in reports now return an error message, rather then segmentation faulting.
3512	Indexed character fields now correctly retrieved for condition **missing**.

Current Restrictions

The following restrictions apply for this release on all hardware platforms, except where noted:

- Equality comparisons between floating point data types and scaled integer or numeric string data types may not produce the expected results. Some scaled decimal numbers cannot be represented exactly in floating point, and some floating point numbers have no exact scale decimal representation. Conversions between the two types truncate repeating fractional portions. This is a particular problem when single and double precision values are mixed in an equality comparison, because the different amounts of truncation will cause apparently equal numbers to return a value of not equal. To reduce the occurrence of this problem, when practical, you should store numbers that will be used in equality comparisons as double rather than single precision floating point.
- Abort codes for triggers cannot be greater than 255. Version 2.*n* triggers with abort codes greater than 255 will execute in Version 3.*n*, but the abort code that is returned in the status vector will be the low-order byte of the defined trigger.
- DSQL has a limit of 32K for the length of both queries and data.
- If you copy a relation that includes a computed field, InterBase does not allow you to delete either the new or the original relation due to dependency checking. To delete the relation, first delete the computed field(s), then delete the relation.
- For VMS only, if you have programs built under Version 2.*n* that use a **gds_$dsql_finish** call, you must relink these programs under Version 3.*n*. If you do not relink, the behavior of the programs may be unpredictable.
- In the Apollo multi-threaded server environment, running a program which accesses a view that has been previously deleted will cause an error.
- Remote events via DECnet are supported only for VMS to VMS connections. If you attempt a remote event via DECnet between VMS and Ultrix or between Ultrix and Ultrix, you will see one of the following messages:

```
I/O ERROR DURING "READ END-OF-FILE" OPERATION FOR FILE "DECNET
CONNECTION"

I/O ERROR DURING "SYS$QIO/IO$READVBLK" OPERATION FOR FILE
"DECNET CONNECTION"

SEGMENTATION FAULT
```

- Support for the bridge between InterBase Version 2.*n* and 3.*n* on the Apollo DN10000 is no longer provided. If you are installing InterBase Version 3.2 on the DN10000 and you have databases created in InterBase Version 2.*n*, you must back up the databases before you install InterBase Version 3.2. On the Apollo Domain DN3xxx and DN4xxx, support for the bridge between InterBase Version 2.*n* and 3.*n* is no longer provided.
- The C **pause** () function on Apollo suspends a process and all of its threads that execute the pause. This prevents InterBase from downgrading locks that the **pause** program holds and can cause the database to lock. If you encounter this problem using InterBase on Apollo, you should purge your programs of **pause** functions and use an alternative function. For example, in some cases, the **pause** function can be replaced with an equivalent **sleep** function or you can write a routine which implements **pause** as **sleep**.
- Domain (Verdix) ADA programs that try to catch floating point exceptions may get kernel errors that report unexpected signal faults. If you have this problem, call InterBase Customer Support.
- The Apollo DN10000 has not been certified to work with Apollo's Domain ADA (Verdix). If you encounter any difficulties using Apollo's ADA with InterBase, contact InterBase Customer Support.
- On InterBase Version 3.2 for the Sun SPARC, when running large numbers of database-accessing programs which are linked against the pipe server (**gds_a**) on a machine with limited memory, you may encounter failures resulting in segmentation violations or other errors. The workaround for this is to use the shared library (gdslib), which will also improve performance.
- If you are using VAX Ultrix with disked clients, you must obtain and install an NFS File Locking patch from DEC before InterBase will run correctly. This patch must be installed on both the disked client and the server. The patch includes these files:

  lockd
  gfs_namei.o
  gfs_sysquota.o
  nfs_server.o
  nfs_vnodeops.o
  ufs_gnodeops.o
  ufs_nemei.o
  ufs_syscalls.o
  vnodeops_gfs.o

VERSION 3.1 RELEASE NOTES

Overview

This document describes changes that have occurred since the InterBase Version 3.0 release.

- New features for InterBase Version 3.1
- New InterBase platform support
- Software restrictions and suggested workarounds, when available
- Bug fixes

Version 3.1 Features

New Platforms

Platform support in InterBase Version 3.1 is provided for:

- AViiON RISC port running DG-UX Version 4.3
- Silicon Graphics port running IRIX Version 3.3
- IBM RS/6000 port running AIX Version 3.1
- HP 9000/400 ports running HP-UX Version 7.3 and SR10.3.

New Call Interface Names

To accommodate systems that do not allow dollar signs ($) in call names, InterBase V3.1 now supports call entry points that begin with **isc**. You do not need to update previous call entry points that begin with **gds_$** because InterBase will continue to support them, but it is recommended that you use the **isc** format for any new calls. For example, either of the following calls will cancel an event:

```
gds_$cancel_events(status_vector,db_handle,event_id)

isc_cancel_events(status_vector,db_handle,event_id)
```

Proxy Account Support

On all InterBase V3.1 platforms when you use TCP/IP to connect to a remote database, you attach to the database via a remote server (*gds_inet_server*). The remote server uses either your current user name or a proxy account name that exists in a proxy file on that machine. The proxy file (*/etc/gds_proxy* on UNIX or *sys$manager: gds_proxy.dat* on VMS) associates remote node names and user names with host account names. To allow remote access, you edit the proxy file. The format for the lines in a proxy file is:

```
<remote node name>:<remote user name> <host account name>
```

Proxy file lines cannot have any spaces between the *<remote node name>* and the *<remote user name>*. Each entry must be on a separate line in the proxy file. A wildcard (*) is allowed in place of the remote node names and/or the remote user names in the proxy file.

The order in which you list entries in the proxy file is important because the proxy file is scanned until a match is found. If a wildcard match occurs before an exact match, the wildcard match is used. For this reason you should place wildcard entries at the end of the proxy file. If a line in your proxy file has a wildcard for both the remote node name and the remote user name, all unmatched users will default to the account name associated with that line. For example, if you have this line in your proxy file:

```
* : *  guest
```

any username that has not been matched by a previous line in the proxy file will be logged into the machine as "guest." An existing username account is used only if that username is matched neither exactly nor with a wildcard in the proxy file.

How Proxy Files are Used in Remote Logins

When a remote node connects to and requests a database on another node, the remote node name is validated. Then, to validate the remote user name, the proxy file is scanned for a matching node/user name. If a match is found in the file, the remote node/ user name is mapped to the host account name in the file. If no match is found in the proxy file, the remote user name is used as an account name. Then the account name is validated to assure that it exists on the host machine. Once the node and account names are validated and the database file permissions checked, the database file is opened and attached. You should note that when the multi-client inet server opens the database for the first user, subsequent attachments by other users to that database do not recheck the database file permissions.

Array Processing Using get_slice and put_slice

With two new GDML statements, **get_slice** and **put_slice**, you can retrieve and write an entire array or a selected portion of an array. You must know the array indexes of the items you want to process, and the array to which you write must be large enough to hold the data.

Get_slice, which is used within a **for** loop, retrieves data from an existing database array and writes it to your target array. Its format is:

```
GET_SLICE <context variable>.<array-field-name> [target-
    dimensions] INTO <target_array>
```

Put_slice, which is used as part of a **for** loop that modifies or stores data, retrieves data from your target array and writes it to an array field in your database. Its format is:

```
PUT_SLICE <context variable>.<array-field-name> [target-
    dimensions] FROM <target-array>
```

With either **get_slice** or **put_slice**, you must use a context variable to identify the field you are retrieving or writing.

The separator between the upper and lower bounds of a range in a **get_slice** or **put_slice** statement is a colon (:) rather than two periods. For example, to specify the range 1..5, you use 1:5. Array ranges in host language statements should use the host language range format.

If the array dimensions in a **get_slice** or **put_slice** statement exactly match the dimensions of the database array, the entire array is retrieved or written. To retrieve a single element, you specify single values for each dimension. To retrieve one column or row, you specify one dimension as a range and the other dimension(s) as single values. To retrieve a "rectangular" portion of the array, you would specify, for two or more dimensions, dimension ranges that are smaller than the full dimensions of the target array. For example, if you have the following array, which has the dimensions [1..6, 1..3]:

	1	*2*	*3*	*4*	*5*	*6*
1	x	x	x	x	x	x
2	x	x	x	x	x	x
3	x	x	x	x	x	x

and you wanted to retrieve the underlined values, you would specify [2:5, 2:3] as the dimensions in the **get_slice**. (This column-major order, [1..6, 1..3] and [2:5, 2:3], ap-

plies to FORTRAN. All other languages use row-major order, which, in this case, would be [1..3, 1..6] and [2:3, 2:5].)

The following is a sample Pascal program that uses **get_slice** and **put_slice**:

```
program array_store (input, output);
database db = filename 'array.gdb';
(*=========================================
 the array.gdb file contains the following information:
      define database "array.gdb"
         page_size 1024;
         define field S short(10,2,3);
      define relation TEST_RELATION
         S   position 0,
==========================================*)

var
    i, j, k : integer;
    target_array : array [1..5, 1..2, 1..3] of integer;
begin
ready;
start_transaction;

(* store a whole array *)
store x in test_relation using
   for i := 1 to 10 do
      for j := 1 to 2 do
         for k := 1 to 3 do
            x.s[i,j,k] := i+j*k;
end_store;

(* look at the whole array as seen in the db *)
for x in test_relation
   for i := 1 to 10 do
      for j := 1 to 2 do
         for k := 1 to 3 do
            writeln ('s[', i, ',', j, ',', k, ']=', x.s[i,j,k]);
end_for;

(* copy a piece of the array into user defined array *)
for x in test_relation
   get_slice x.s [1:5, 1:2, 1:3] into target_array;
end_for;

(* display the user's array, then change the contents *)
```

```
for i := 1 to 5 do
   for j := 1 to 2 do
      for k := 1 to 3 do
      begin
   writeln ('test[', i, ',', j, ',', k, ']=',
         target_array[i,j,k]);
      target_array[i,j,k] :=  100 * i + 10 * j + k;
    end;

(* use put_slice to change part of the database array *)
for x in test_relation modify x using
   put_slice x.s [3:5, 1:2, 1:3] from target_array;
end_modify;
end_for;

(* see what that did to the array in the database *)
for x in test_relation
   for i := 1 to 10 do
      for j := 1 to 2 do
         for k := 1 to 3 do
         writeln ('s[', i, ',', j, ',', k, ']=', x.s[i,j,k]);
end_for;
commit;
finish;
end.
```

Array Elements in Database Queries

Version 3.1 **qli** allows you to reference array elements in database queries and RSEs. You can use **qli** keywords, such as **print**, **list**, **select**, **where**, and **with**, to reference array elements. You cannot store or modify array elements in **qli**.

For example, to list the values of the [1,2,1] array element for each record in the ARRFIELD field of relation TEST, you would use this query in **qli**:

```
for test list arrfield[1,2,1]
```

To select and print array values that are equal to "2", you would use:

```
for test with arrfield[1,2,1]=2 print arrfield[1,2,1]
```

User Defined Functions for Blobs and Arrays

In Version 3.1 you can create UDFs (user defined functions) to which you can pass blobs and/or arrays and from which blobs can be returned. These UDFs are defined like all other UDFs (see Chapter 9 of the *Data Definition Guide*), except that you pass a blob by reference to a blob UDF structure, and you pass an array by reference to a scalar-array-descriptor, which will be defined in the *Array UDFs* section below.

A pointer to a structure, not the actual blob or array data, is passed to the UDF. The UDF does not open or close the blob, but invokes those functions from the control structure. The structure, which differs for blobs and arrays, is described as used in the C language, in the sections below.

Blob UDFs

When a blob is passed by reference, the structure to which it points is:

```
typedef struct blob {
    short    (*blob_get_segment)();
    int      *blob_handle;
    long     blob_number_segments
    long     blob_max_segment
    long     blob_total_length
    void     (*blob_put_segment)();
    };
```

The above typedef declaration must be included in all blob UDFs. The fields in the blob typedef structure are:

- **blob_get_segment**: Pointer to a function that is invoked to read a segment from the blob (if the blob is being passed to the UDF). As arguments, this function takes the blob handle, the address of a buffer into which to place the data, the size of that buffer, and the address of the variable into which the size of the actually read data is placed.
- **blob_handle**: The handle of the blob (whether the blob is passed to the UDF or returned from the UDF).
- **blob_number_segments**: The total number of segments in the blob (if the blob is being passed to the UDF).
- **blob_max_segment**: The size of the largest segment in the blob (if the blob is being passed to the UDF).

- **blob_total_length**: The total size of the blob (if the blob is being passed to the UDF).
- **blob_put_segment**: Pointer to a function to be invoked to write a segment to the blob (if the blob is being returned from the UDF). As arguments, this function takes the blob handle, the address of a buffer containing the data to be written, and the length of the data.

NOTE

A UDF that returns a blob is not written as a function. Instead of returning the structure describing the output blob, that structure is passed in as the m^{th} +1 parameter, where m is the number of parameters declared for the function.

The following example is a UDF that takes two blobs as input and returns one new blob, which is a concatenation of the two input blobs:

```
/*============================================================
Function definition -- how you define this UDF in gdef
define function blob_concatenate
    entry_point 'blob_concatenate'
    blob by reference,
    blob by reference,
    blob by reference return_argument;
============================================================*
 * Blob passing structure */
typedef struct blob {
   short       (*blob_get_segment)();
   int         *blob_handle;
   long        blob_number_segments;
   long        blob_max_segment;
   long        blob_total_length;
   void        (*blob_put_segment)();
} *BLOB;

extern char *gds_$alloc();
#define MAX(a, b) (a > b) ? a : b

#define DELIMITER "----------------------------------------"
/* NOTE:  Although the function is defined to return a blob
   by reference, you always write the function with the last
   argument being the blob you wish to return */
```

```
blob_concatenate (from1, from2, to)
    BLOB        from1, from2, to;
{
char    *buffer;
short   length, b_length;

b_length = MAX (from1->blob_max_segment, from2-
>blob_max_segment);
/* use gds_$alloc to allocate memory for the buffer */
buffer = gds_$alloc (b_length);

 * write the first blob into the return blob */
while ((*from1->blob_get_segment)(from1->blob_handle, buffer,
          b_length, &length))
    (*to->blob_put_segment) (to->blob_handle, buffer, length);

 * now write the delimiter line */
(*to->blob_put_segment) (to->blob_handle, DELIMITER, sizeof
(DELIMITER) - 1);

 * and finally, write the second blob *
while ((*from2->blob_get_segment)(from2->blob_handle, buffer,
          b_length, &length))
    (*to->blob_put_segment) (to->blob_handle, buffer, length);
 * use gds_$free to release the allocated memory *
gds_$free (buffer);
}
```

Array UDFs

Arrays can be passed to a UDF, but they cannot be returned from a UDF. Arrays passed to UDFs are multidimensional arrays of a uniform, scalar data type. When a array is passed, the control structure to which it points is:

```
typedef struct sad {
    char    dsc_dtype;
    char    dsc_scale;
    short   dsc_length;
    long    *dsc_address;
    long    dsc_dimensions;
    struct  dsc_repeat
```

```
{
long     dsc_lower;
long     dsc_upper;
}dsc_rpt [1];
```

The above typedef declaration must be included in all Array UDFs.

The fields in the array typedef structure are:

- **dsc_dtype**: A type code that indicates the type of data in the array. Valid types and their associated codes are:

type text	1
type cstring	2
type varying	3
type short	4
type long	5
type quad	6
type real	7
type double	8
type date	9
type blob	10
type d_float	12

- **dsc_scale**: If the **dsc_dtype** is a short or long integer, **dsc_scale**, which may be 0, must also be supplied.
- **dsc_length**: The length of a single element of the array.
- **dsc_address**: The pointer to the actual array data.
- **dsc_dimensions**: The number of dimensions in the array.
- **dsc_upper/dsc_lower**: The upper and lower dimension limits for each dimension.

An example of an array UDF which returns the numeric average of the values in an array is:

```
/*============================================================
Function definition -- how you define this UDF in gdef
define function long_array_average
    module_name 'FUNCLIB'
    entry_point "long_array_average"
    long by scalar_array_descriptor,
    long by value return_value;
define relation r
```

```
    number short,
    array long (10, 2),
    long_average long computed by (long_array_average (array));
============================================================*/
typedef struct sad {
    char        dsc_dtype;
    char        dsc_scale;
    short       dsc_length;
    long        *dsc_address;
    long        dsc_dimensions;
    struct      dsc_repeat
    {
    long        dsc_lower;
    long        dsc_upper;
    }           dsc_rpt [1];
} *DSC;

long long_array_average (desc
    DSC         desc;
{
   long    n, total, avg, *ptr, *end;
   n = elements (desc);
   total = 0;
   for (ptr = desc->dsc_address, end = ptr + n; ptr < end; ptr++)
    total += *ptr;
   avg = total / n;
   return avg;
}

static elements (desc)
    DSC         desc;
{
   long           count, i;
   struct dsc_repeat    *tail, *end;
   count = 1;
   i = 0;
   for (tail = desc->dsc_rpt, end = tail + desc->dsc_dimensions;
       tail < end; tail++)
          count *= tail->dsc_upper - tail->dsc_lower + 1;
return count;
}
```

Support for D_FLOAT Format

In the VAX/VMS environment, D_FLOAT double-precision data from a user application can be stored in an InterBase database in G_FLOAT format. A D_FLOAT flag (d_float) within **gpre**, specified at program compile time, determines how double-precision data will be passed between a user's application and an InterBase database. If the D_FLOAT flag is specified, then double-precision data passed from the user's application will be assumed to be in D_FLOAT format and stored within an InterBase database in G_FLOAT format. Data comparisons within a database will be performed in G_FLOAT format. Data returned to the user's application from an InterBase database will be returned in D_FLOAT format.

Support for HP-UX Cluster Configuration

InterBase Version 3.1 supports HP9000/300 cluster configurations. It recognizes cnodes and coordinates database access through the server (root node).

ALSYS ADA Support

In InterBase, V3.1C support for ALSYS Version 5.1.1 on Apollo SR10 will be updated to support Version 5.2, where an integer is defined as 32 bits instead of the previous 16-bit definition.

Apollo C++ Support

InterBase provides support on Apollo for the Apollo-provided C++. Apollo C++ input files with embedded GDML or SQL should have file extension "exx". The output file produced by **gpre** when you preprocess your file will have the extension "cxx". If you use the **gpre** option **-cxx**, the input file extensions are not required.

V3.1 Bug Fixes

NIST SQL Compliance Bug Fixes

1016 Arithmetic expressions that contain aggregate functions now parse correctly.
1018 (same as 1016 above).
1502 Comparison subquery that selects no records now executes correctly.
1517 Nested **group by** with **having** now works correctly.
1518 Precedence of unary negation operator has been corrected.
1541 Cursor now must be successfully closed before executing **open cursor** command.
1542 (same as 1541 above).
1544 **union all** - optional word **all** now accepted.

General Bug Fixes

384 Expressions now can be listed (**qli**).
415 Expression with global aggregates can now be selected (**qli**).
416 (same as 415 above).
832 **select * group by** now works correctly (**qli**).
837 ADA exception can test for EOF error with 'end-error' condition.
957 SQL **fetch** after close cursor now returns an error.
1532 PL/1 **based** is no longer assumed to be **based on.**
1595 SQL insert from sub-query containing **distinct** now works correctly (**qli**).
1691 (same as 832 above).
2864 (same a 415 above).
2891 Optional word ALL as argument to aggregate function is now accepted.
2898 Yacc/Lex symbols now not visible to user programs.
2930 (same as 832 above).
2936 (same as 832 above).
2977 Number of relations that can be joined has been increased.
3002 **gcsu** and **gds_cserver** now accept slashes (/) on switches (VMS).
3009 Defined query headers now used for **list.**
3010 Remote DECnet events now working correctly.
3040 Maximum size of DSQL query has been increased.
3047 Sun NFS now correctly checks for local/remote directories.

3059 **gfix -s** now works correctly for databases with filenames of less than 5 characters.

3061 **gpre** now generates valid PL/1 code when run with the **/raw** switch.

3062 External files ok under V2.5 now work under V3.*n* (VMS).

3097 **qli** now accepts <> as comparison operator.

3102 Wollongong error messages have improved (VMS).

3104 Indexes with missing information are reported during **gbak**.

3106 **gbak -l** now rolls back limbo transactions when backup is restored.

3107 **gbak -y** or **/y** works correctly, but must be used with an argument.

3112 DECnet - excessive use of NCP links corrected.

3115 **at end/end-fetch** clause now generates code that works correctly in COBOL.

3127 Descending multi-key indexes used in partial key search now retrieve all values .

3131 New Ethernet controller (NE0) now is supported for MIPS Ultrix.

3146 **blp_prot_mask** now returns the Protect bit.

3151 Retrieving based on partial retrieval in multiple multi-segment indexes does not cause an access violation (VMS).

3225 Retrievals from an indexed field return null records after non-null records.

3250 **gfix** l-sweep and sweep dpb attach parameter correctly garbage-collect.

- On some V3.1B platforms, **iscinstall** will not allow you to enter a non-default Ethernet ID. If you use the default ID by pressing RETURN, it will accept the default ID. If you are registering a node other than the current node (for example, a diskless client), then you should manually enter into *isc_license.dat*, the registration information provided by InterBase Technical Support using a text editor.
- Support for *gds.authorize* will end with InterBase Version 3.2.

VERSION 3.0 RELEASE NOTES

Overview

This document:

- Lists new features for InterBase Version 3.0
- Describes Version 2.*n* to Version 3.0 compatibility issues
- Explains changes in component behavior from Version 2.*n* to Version 3.0
- Describes omissions and corrections to the Version 3.0 documentation set
- Describes software restrictions and suggested workarounds when available

WARNING

If you are running InterBase on SunOS 3 or SunOS 4, have databases created with a previous version of InterBase, and used *gds_b.a* to statically link to your application (**-lgds_b**), you must relink after you install Version 3.0. **Databases may be corrupted if you do not relink**. If you originally linked using the dynamic library (**-lgdslib**) or the pipe server (**-lgds**) you do not need to relink.

If you are using a SunOS 3/470, contact your sales representative before attempting to install InterBase.

Version 3.0 Features

The following features are new in Version 3.0:

- Multi-client TCP server
- Multi-threaded Apollo mailbox server
- UNIX journaling
- Central server
- Database shadowing
- Automatic multi-database transaction recovery
- Multiple triggers per relation
- User-defined functions
- Multi-dimensional arrays

- Event alerters
- Blob filters
- SQL metadata support in **qli** and DSQL
- SQL **grant/revoke** security statements
- New **qli** statements:
  - **show filter**
  - **show filters**
  - **show function**
  - **show functions**
- **pyxis** enhancements, such as support for dynamic menus
- GDML enhancements:
  - **commit** command/**commit** statement
  - case-insensitive sort capability
  - **matching using** condition
- Performance improvements
- New platform support for:
  - HP 9000 Series 800
  - HP 9000 Series 300
  - HP 3000 running MPE XL
  - Ultrix DECstations and DECsystems
  - SCO Xenix
  - SCO UNIX

Version 2.*n* to Version 3.0 Compatibility

The following sections describe compatibility issues to consider if you plan to run databases created using InterBase Version 2.5 (or earlier) with Version 3.0.

Changes to the On Disk Structure

The On Disk Structure (ODS) is significantly revised in Version 3.0 and now supports more complex functions and provides better performance. InterBase Version 3.0 is compatible with and can access both the Version 2.*n* ODS (ODS version 4) and the Version 3.0 ODS (ODS version 6). To run a Version 2.*n* database with Version 3.0, you use the bridge, which is described below. To use the new Version 3.0 functions, you must run the Version 3.0 **gbak** command to back up and restore your existing database(s).

NOTE

If you are a VAX Ultrix or HP9000 Series 800 user, you must back up all Version 2.*n* databases using Version 2.*n* **gbak** *before* installing the new version of InterBase or you will not be able to access those databases.

If you are an HP9000 Series 300 or 600, or DECstation 2100, 3100, or 5000 Series user and have a beta copy of Version 3.0 installed on your system, you must back up all Version 3.0 databases using the Version 3.0 beta **gbak** *before* installing a new copy of Version 3.0 or you will not be able to access those databases.

The Bridge between Version 3.0 and Version 2.*n*

You can access a Version 2.*n* database through Version 3.0 with a modified copy of the Version 2.5 access method called the bridge. The bridge is not available for the HP 9000 Series 300, 600 or 800, the HP 3000, or for any system running VAX Ultrix. (This section uses **qli** to describe access to the bridge through Version 3.0.)

If you define a database using Version 3.0, the database is a Version 3.0 ODS 6 database. For example, the following **qli** statements show version information for a local Version 3.0 database (*mydata.gdb*), and a remote database (*v3_ods6.gdb*):

```
    (lfg4e)"
    InterBase/apollo (remote server), version "AX-V3.0F/mbx
    (fecb)"QLI> show version
    QLI, version "AX-V3.0F"
QLI> define database mydata.gdb
```

```
QLI> show version
    QLI, version "AX-V3.0F"
        Version(s) for database "mydata.gdb"
        InterBase/apollo (access method), version "AX-V3.0C"

QLI> finish
QLI> ready v3_ods6.gdb
QLI> show version
    QLI, version "AX-V3.0F"
    Version(s) for database "v3_ods6.gdb"
    InterBase/apollo (access method), version "AX-V3.0F mbx
```

You can access Version 2.*n* and Version 3.0 databases through the bridge on Version 3.0. You cannot access a Version 3.0 database directly through Version 2.*n*, but you can access a Version 3.0 database remotely (so that your Version 2.*n* remote interface is accessing the Version 3.0 remote server). If you access a Version 3.0 database through Version 2.*n*, InterBase returns the following message:

```
QLI> show version
QLI> version "AX-V2.5B"
QLI> ready v3_ods6.gdb
*** QLI error from database "v3_ods6.gdb" ***
unsupported on disk structure for file v3_ods6.gdb; found 0,
support 4
```

The found ODS number, 0, as shown in the message above, is incorrect. Although you have reached an ODS 6 database, the Version 2.*n* access method does not recognize it as an identifiable ODS. The support number, 4, which is the ODS for Version 2.*n*, is reported correctly by **qli.**

If you try to use Version 3.0 functions with a Version 2.*n* database, you may receive messages of the form:

```
database version is too old for <new functionality>: use GBAK
first
```

or

```
database version is too old for new syntax:
<syntax/functionality>
```

Use **gbak** to backup and restore your database to ODS 6 if possible. This process is described in the Version 3.0 installation notes.

The Apollo Bridge

On the Apollo, the bridge is an image in the directory */interbase/lib*. In the following example, a Version 2.*n* database is accessed through Version 3.0 and the bridge is identified as Version 2.5B.

```
QLI> ready v2_ods4.gdb
QLI> show version
    QLI, version "AX-V3.0F"
    Version(s) for database "v2_ods4.gdb"
        InterBase/apollo (access method), version "AX-V2.5U"
```

On an Apollo you may receive the following message:

```
QLI> show version
QLI> version "AX-V3.0F"
QLI> ready v3_ods6.gdb
** QLI error from database "v3_ods6.gdb" **
unsupported on disk structure for file v3_ods6.gdb; found 0,
support 4
```

The above message indicates that your Version 3.0 database is being accessed by a Version 2.*n* **gds_server**. This may be the result of an aborted installation that left a Version 2.*n* server running on a Version 3.0 node. To resolve this conflict, determine which node is locking the database by performing these two steps:

1. Kill the **gds_server** and the **gds_guardian** on that node.

2. Start a new Version 3.0 **gds_server**.

If the server is still a Version 2.*n* server, check that the installation on that node was not aborted, which might have left a Version 2.*n* server in the */interbase/com* subdirectory.

The Sun Bridge

On the Sun, the bridge is implemented through the pipe server as **gds_pipe** in */usr/interbase/bin*. The bridge is built into the **gds_inet_server**, so the remote server can access both Version 2.*n* and Version 3.0 databases. If you access a Version 2.*n* database through Version 3.0, you see the following message:

```
QLI> ready v2_ods4.gdb
QLI> show version
    QLI, version "S3-V3.0F"
    Version(s) for database "v2_ods4.gdb"
        InterBase/sun (access method), version "S3-V2.5U"
        InterBase/sun (pipe interface), version "S3-V3.0F"
```

The VMS Bridge

On VMS, the bridge is implemented through a second shareable library, *gdsshr.exe*, which appears as shown below. Both the DECnet and inet servers use the bridge, so they can access both Version 2.*n* and Version 3.0 databases.

```
QLI> ready v2_ods4.gdb
QLI> show version
    QLI, version "VM-V3.0FV3.0F"
    Version(s) for database "v2_ods4.gdb"
        InterBase/vms (access method), version "VM-V2.5U"
```

Bridge Restrictions

You cannot update the metadata on any platform for a Version 2.*n* database through the bridge. Switch to Version 2.*n* if you want to change the metadata for a Version 2.*n* database, or use **gbak** to backup and restore your databases to Version 3.0.

Changes to InterBase Components

The following section describes changes to features from InterBase Version 2.*n* to Version 3.0.

Changes to qli

- Using the **show relations** or **show procedures** statements in **qli** may produce a slightly different display than in Version 2.*n*. The display is formatted in multicolumns based on:
  - — The value supplied for the **set columns** command in your **qli** start-up file.
  - — The default column width. (On Apollos, the default column width corresponds to the current window width.)
- If you use an unsigned edit string to retrieve a negative number in **qli**, you now get a data overflow. In Version 2.*n*, InterBase returned the absolute value of the number.

  To retrieve the absolute value of a number, use the ABS user-defined function, included in the */usr/interbase/examples* directory. For example:

```
QLI> print abs(-3);
 3

QLI> declare foo double;
QLI> foo = -3;
QLI> print abs(foo);
 3
```

Changes to Forms

Field editing in **fred** is different in this release. If you are in a form, pressing the Tab key places the cursor on the next field, rather than the next editable field.

Two form modification modes are available: navigation mode and edit mode. You use navigation mode to move from field to field; you use edit mode to change modifiable fields. **fred** always starts in navigation mode, where you can use the arrow keys, Return key, Tab key, or Delete key to move from field to field. To shift into edit mode, move the cursor to a modifiable field and press the key that corresponds to Edit, Insert-Overstrike, Erase, or Insert (see the list below). To return to navigation mode, either press the Edit key or press any key that is not a field editing key (i.e., arrow, Tab, Return, or a function key).

NOTE

If you are using an HP machine and you are not using VT100 emulation mode, you must use the HOME key instead of the ENTER key.

FUNCTION	DESCRIPTION	KEY ON APOLLO	KEY ON ALL OTHERS*
Edit	Toggles between edit mode and navigation mode	EDIT	Ctrl-G
Right	Moves cursor one character to right	Right Arrow	Right Arrow
Left	Moves cursor one character to left	Left Arrow	Left Arrow
Delete	Deletes character to left of cursor	BACKSPACE	Delete
Delete-Next	Deletes current character	CHAR DEL	Ctrl-F
Go-To-Start	Moves cursor to first position of field	Left Bar Arrow	Ctrl-H
Go-To-End	Moves cursor to last position of field	Right Bar Arrow	Ctrl-E
Insert-Overstrike	Toggles between insert and overstrike modes	INS	Ctrl-A
Erase	Deletes contents of entire field	LINE DEL	Ctrl-U
Insert	Inserts any printable character into field	Any character	Any character

*Forms are not available on the HP 3000.

Using the Mouse

For Apollo workstations, you can use the mouse to navigate in forms and menus. The left mouse button corresponds to the ENTER key and the right button corresponds to the RETURN key. For horizontal and vertical menus, moving the mouse moves the menu cursor to the next or prior menu choice. You can also use the mouse to move the cursor among and within fields on a form.

Changes to SQL

The way in which SQL indicator variables work has changed:

- If the SQL indicator variable contains a negative number, the value of the associated field is presumed missing.
- If the variable contains 0 or a number greater than 0, the value of the associated field is presumed present.

Changes to DSQL

- DSQL may report errors that more input parameters are required for a command than the given SQLDA can provide. Previously, you set SQLN to the number of variables the SQLDA allows, and set the SQLD to the number of variables used. Previous releases of InterBase did not check SQLD, which resulted in access violations if users tried to access a variable that was not set up.

  Version 3.0 checks the variable number set by SQLD and returns an error if it tries to access a higher variable number. If you have to run a program that does not set the SQLD in the input SQLDA, you must change the code to adhere to this requirement.
- For ADA, the declaration and use of SQLDA structures have changed in Version 3.0. See Chapter 4 of the *DSQL Programmer's Guide* for information.
- DSQL data types and the constants used to produce them have changed in Version 3.0. Two new constants, SQL_FLOAT and SQL_FLOAT + 1, produce 4-byte fields of data type float.

  If you have programs set up to retrieve float type data as double, DSQL now returns the data as float. You can do either of the following:

  — Adjust the program to accept float data.

  — Reset the SQLDA to indicate a data type of SQL_DOUBLE instead of SQL_FLOAT. You must reset the SQLDA after you issue a **prepare** or **describe** statement
- The DSQL indicator variable, SQLIND, now works as follows:

  — If DSQL_IND is a negative number, the value of the associated field is missing.

  — If DSQL_IND is equal to or greater than 0, the value of the associated field is present.

Index

V

INTERBASE

BORLAND

CORPORATE HEADQUARTERS: BORLAND INTERNATIONAL INC., 1800 GREEN HILLS ROAD, P. O. BOX 660001, SCOTTS VALLEY, CA 95067-0001, (408) 438-5300. OFFICES IN: AUSTRALIA, DENMARK, FRANCE, GERMANY, ITALY, JAPAN, NEW ZEALAND, SINGAPORE, SWEDEN, TAIWAN, AND UNITED KINGDOM. PART # INT0033WW21690