

InterBase 6

# Getting Started



**Borland®**

SOFTWARE CORPORATION

100 Enterprise Way, Scotts Valley, CA 95066 <http://www.borland.com>

Borland Software Corporation may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not convey any license to these patents.

Copyright 2001 BorlandSoftware Corporation. All rights reserved. All InterBase and Borland products are trademarks or registered trademarks of BorlandSoftware Corporation. Other brand and product names are trademarks or registered trademarks of their respective holders.

1INT0060WW21000 21006

# Table of Contents

CHAPTER 1	<b>System Requirements</b>	
	Disk space requirements . . . . .	5
	Windows system requirements . . . . .	6
	UNIX system requirements . . . . .	6
	Linux . . . . .	6
	Solaris . . . . .	6
	Requirements for other platforms . . . . .	6
CHAPTER 2	<b>Installation</b>	
	Installing InterBase on Windows . . . . .	7
	Installing InterBase on Windows. . . . .	8
	Configuring InterBase . . . . .	8
	Logging on . . . . .	8
	Uninstalling InterBase on Windows . . . . .	9
	Installation on UNIX . . . . .	9
	Preparing to install on UNIX . . . . .	10
	Installing on Solaris . . . . .	10
	Installing on Linux. . . . .	13
	Running multiple versions on UNIX . . . . .	16
	Uninstalling InterBase on UNIX . . . . .	16
	Installing and accessing InterBase documentation . . . . .	17
	Full-text searching . . . . .	17
	Links. . . . .	18
	Installing Acrobat Reader With Search . . . . .	18
CHAPTER 3	<b>Migrating to InterBase 6</b>	
	Migration process . . . . .	19
	Migration Issues . . . . .	20
	InterBase 6 SQL dialects . . . . .	20
	Servers, databases, and ODS 10 . . . . .	20
	Clients and databases . . . . .	21

Keywords used as identifiers . . . . .	21
Understanding SQL dialects. . . . .	22
Setting SQL dialects . . . . .	23
Setting isql client dialect . . . . .	23
Setting the gpre client dialect. . . . .	24
Setting the gfix client dialect . . . . .	25
New features . . . . .	25
Features available in all dialects . . . . .	25
Features available only in dialect 3 databases . . . . .	26
New keywords . . . . .	27
Delimited identifiers. . . . .	28
DATE, TIME, and TIMESTAMP datatypes . . . . .	29
DECIMAL and NUMERIC datatypes . . . . .	37
Compiled objects . . . . .	38
Generators . . . . .	39
Miscellaneous issues. . . . .	39
Migrating servers and databases . . . . .	40
“In-place” server migration. . . . .	40
Migrating to a new server . . . . .	42
About InterBase 6, dialect 1 databases . . . . .	43
Migrating databases to dialect 3 . . . . .	43
Overview . . . . .	43
Method one: in-place migration . . . . .	44
Method two: migrating to a new database . . . . .	49
Migrating clients . . . . .	51
IBReplicator migration issues . . . . .	53
Migrating data from other DBMS products . . . . .	53

# System Requirements

InterBase Server runs on a variety of platforms, including Microsoft Windows NT 4.0, Windows 2000, Windows 98/ME, and several UNIX/Linux operating systems.

The InterBase server software makes efficient use of system resources on the server node; the server process uses little more than 1.9MB of memory. Typically, each client connection to the server adds approximately 115KB of memory. This varies based on the nature of the client applications and the database design, so this figure is only a baseline for comparison.

The InterBase client also runs on any of these operating systems. In addition, InterBase provides the InterClient Java client interface using the JDBC standard for database connectivity. Client applications written in Java can run on any client platform that supports Java, even if InterBase does not explicitly list it among its supported platforms. Examples include Macintosh and Internet appliances with embedded Java capabilities.

---

## Disk space requirements

InterBase requires 35 MB for a full install that includes InterBase Server and Client, Adobe Acrobat Reader, and the full document set. 17MB for InterBase Server and Client alone.

During operation, InterBase's sorting routine requires additional disk space as scratch space. The amount of space depends on the volume and type of data the server is requested to sort.

---

## Windows system requirements

**Operating system:** Windows NT 4.0 with Service Pack 6a, Windows 98, Windows ME, or Windows 2000 Service Pack 1

**Memory:** 16 megabytes minimum; 64 recommended for a server

**Processor/Hardware model:** 486DX2 66MHz minimum; Pentium 100MHz or greater recommended for a multiclient server

**Compiler:** Microsoft Visual C++ 4.2 (or later) or Borland C++ Builder 4.0 (or later)

---

## UNIX system requirements

---

### Linux

**Operating system:** RedHat 6.2 or 7.0; TurboLinux 6.0; SuSe 7.0; Mandrake 7.2

**Memory:** 32 megabytes minimum; 64 recommended for a server

**Processor/Hardware Model:** Intel 486 or higher

**Compiler:** GCC or G++

---

### Solaris

**Operating system:** Solaris 2.6 or 2.7

**Memory:** 32 megabytes minimum; 64 recommended for a server

**Processor/Hardware Model:** SPARC or UltraSPARC

**C Compiler:** SPARCWorks SC 4.2 C compiler

**C++ Compiler:** SPARCWorks SC 3.0.1. C++ compiler

**Note** InterBase on Solaris uses a OS-specific install utility called **pkgadd**. Install the Solaris software using the **setup.ksh** script.

---

## Requirements for other platforms

Refer to online sources of information, including release notes included on your product media and the Borland web site, for more information on supported platforms, operating systems, and compilers.

# CHAPTER 2 Installation

This chapter contains detailed instructions for installing InterBase on Windows 98/ME, Windows NT, Windows 2000, Linux, and Solaris.

InterBase might install differently on other platforms due to operating system requirements. Refer to installation notes included in the software package, on the media, or on the Borland web site (<http://www.borland.com/interbase>) for platform-specific installation instructions.

---

## Installing InterBase on Windows

By default, InterBase files are installed in **c:\Program Files\InterBase** on Windows platforms. You can use the INTERBASE environment variable to change this location. See the *Operations Guide* for information about setting environment variables.

**IMPORTANT** You cannot install InterBase onto a network (mapped) drive.

---

## Installing InterBase on Windows

1. If you are installing InterBase 6 on a system running Windows NT or Windows 2000, log in as a user with Administrator privileges.
2. If InterBase 5.6 or earlier is present on the system:
  - Backup all InterBase databases. This is a required step.
  - Stop the InterBase server.
  - Use **Control Panel | Add/Remove Programs** to uninstall any previous versions of InterBase.

**Note** Please see the Migration section of this manual for a detailed explanation of the decisions and steps required in migrating databases from InterBase 5 to InterBase 6.

3. If the file **gds32.dll** is present on the hard disk, delete the file; the uninstall step does not remove this file and only one version of this file can exist on the disk.
4. Insert the InterBase 6 distribution CD-ROM into the CD drive.
5. If Autoplay does not start the installation process, run **setup.exe** from the root directory of the CD-ROM.
6. Follow the directions on screen.

---

## Configuring InterBase

You can control many aspects of how InterBase runs, including server and database configurations, backup and restore, and database replication. See the *Operations Guide* for information on these topics.

---

## Logging on

To attach to any database, you must have a user name and password. When you first install InterBase, there is one user defined: SYSDBA. The password is *masterkey*. SYSDBA has special privileges that override normal SQL security, and there are database maintenance tasks that only SYSDBA can perform. You should change the SYSDBA password immediately after installing InterBase.

Use IBConsole or the **gsec** utility to change a password and create additional users. For more information see the IBConsole online help or the *Operations Guide*.



---

## Uninstalling InterBase on Windows

To uninstall InterBase and InterBase Express, use **Control Panel | Add/Remove Programs**. Choose InterBase or InterBase Documentation, and click Add/Remove.

InterBase documentation is removed when the InterBase Client or Server is uninstalled.

- The InterBase Server, Guardian, and InterServer processes must not be running when you uninstall the software. To stop one of these applications, right-click its icon in the Task Tray and selecting Shutdown. To stop one of these services (Windows NT/2000), use the **Control Panel | Services** applet.
- You must be logged in as a user with Administrator privileges to install or uninstall InterBase on Windows NT/2000.
- Uninstall never removes **isc4.gdb** or files created by the server process, including **interbase.log**, **host.evn**, **host.lck**.
- The Windows InterBase installation allows you to choose between performing a complete install or selecting individual components. If you choose to install components at different times, the uninstall program removes only the components selected for the last install.
- The ODBC driver or the ODBC driver manager must be removed manually; there is no uninstall available through the Windows control panel.

---

## Installation on UNIX

This section provides instructions for installing InterBase 6 on Solaris and Linux. For platform-specific instruction on installing InterBase on other UNIX brands, see the InterBase web site (<http://www.borland.com/interbase>).

By default, InterBase files are installed in **/opt/interbase**. On the Solaris platform, the install program still creates a symbolic link from **/usr/interbase** to **/opt/interbase**. On Linux, there is no symbolic link from **/usr/interbase** to an actual install directory. You can use the **INTERBASE** environment variable to change this location. See the *Operations Guide* for information about setting environment variables.

---

## Preparing to install on UNIX

### 1. Save older databases

InterBase 6 uses a new On-Disk Structure (ODS), ODS 10. Databases created with InterBase 5 used ODS 9.1. To take advantage of new InterBase 6 features, you must use **gbak** to back up any databases that you intend to use with the Version 6 software. V5 databases can be backed up using V5 **gbak** or with the InterBase 6 **gbak** if you are performing the backup after the install. Once a database has been converted to ODS 10, it cannot be converted back to earlier versions of the ODS. For more information, refer to the Migration chapter.

### 2. Save customization files

If you have InterBase 5 installed on the server and you want to preserve the customization files, copy them to a safe place, for example:

```
gbak -b /usr/interbase/isc4.gdb /var/tmp/isc4.gbak
cp /usr/interbase/isc_license.dat /var/tmp
cp /usr/interbase/isc_config /var/tmp
```

You can skip this step if you haven't customized these files in a previous installation.

**Note** When you use **pkgrm** remove InterBase V5 for Solaris, these files are automatically saved in **/usr/tmp**.

### 3. Save older versions

If InterBase 5 is running on your server, shut it down. To save the current version, rename the directory, for example:

```
ibmgr -shut -user sysdba -password password
mv /usr/interbase /usr/interbase.save
```

### 4. Point to the install directory

If you install into a directory other than **/opt/interbase** on Solaris, the install program creates a symbolic link from **/usr/interbase** to that directory. If you install into a directory other than **/opt/interbase** on Linux, the install program creates a symbolic link from **/opt/interbase** to that directory.

---

## Installing on Solaris

InterBase requires Solaris versions 2.6.x. or 7. These instructions are for both server and client installations:

1. Log in to your database server as **root**.
2. Load the CD-ROM with the InterBase 6 product. Mount the CD-ROM at an appropriate mount point. If you have the volume manager running, this is done for you, and the CD-ROM mounts according to its label. For instance `/cdrom/interbase_sos_V6` for the CD-ROM of InterBase 6 for Solaris.
3. Run **setup.ksh**. **setup.ksh** brings up the following menu:

```

1.Install InterBase Client and Server software
2.Install InterBase Client Only software
3.Install Adobe Acrobat Reader software
4.Exit
Enter selection.(default 1) [1-4]
```

The following sections describe these options

### ► *Installing InterBase Client and Server on Solaris*

When you select “Install InterBase Client and Server software” from the menu, the setup script displays the following messages and starts the **pkgadd** utility:

```

Starting InterBase Client and Server Install, please wait...
set NONABI_SCRIPTS=TRUE
export NONABI_SCRIPTS
pkgadd -d CDROM_DIR/Interbase6.0_ClientServerpkg
```

The following packages are available:

```

1  interbase InterBase RDBMS Software
    (sparc) InterBase Version 6.0
```

```

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,?,q]:
```

1. Press **Enter** to choose the Install InterBase default.

```

Processing package instance <interbase> from
</cdrom/interbase_SOS_V6>
```

```

InterBase RDBMS Software
(sparc) InterBase Version 6.0
```

```

Enter the absolute pathname of the install directory
(default /usr) [?,q]
```

2. Press **Enter** to accept the default or type in the pathname in which to create the **interbase** directory.

```

Using </usr> as the package base directory.
```

```
## Processing package information.
## Processing system information.
## Verifying disk space requirements.
## Checking for conflicts with packages already installed.
## Checking for setuid/setgid programs.
```

This package contains scripts which will be executed with superuser permission during the process of installing this package.

Do you want to continue with the installation of <interbase> [y,n,?]

3. Choose *y*. The installation script must have superuser privilege to modify system files and create symbolic links.

```
Installing InterBase RDBMS Software as <interbase>
## Installing part 1 of 1.
/usr/interbase/install.txt
/usr/interbase/Release_Notes.pdf
. . .
[ verifying class <none> ]
## Executing postinstall script.
. . .
```

The InterBase install script looks for the InterClient installation script on the CD-ROM, and runs it if possible. See **“Uninstalling InterBase on UNIX” on page 16**.

Thereafter, the install script returns to **pkgadd**:

```
Installation of <interbase> was successful.
The following packages are available:
    1  interbase InterBase RDBMS Software
        (sparc) InterBase Version 6.0

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]:
```

4. Choose *q*. There is only one package to install using **pkgadd**.
5. If you saved customization files, restore them now with the ones you backed up before installing InterBase 6. For example:
 

```
# cp /tmp/isc_config /usr/interbase
```
6. If you intend to run the **ibserver** daemon as a user other than root, create a UNIX user account named “interbase” on your machine. Log in as this user before starting the SuperServer with the **ibmgr** utility.

**Note** We recommend that you create an “interbase” user account and use this account for all database maintenance; this will make tasks such as backup and restore run much more smoothly.

7. Execute the following command to start the InterBase Server:

```
# echo "/usr/interbase/bin/ibmgr -start -forever" | su interbase
```

This starts the SuperServer daemon (**ibserver**) and a guardian (**ibguard**) program that keeps track of **ibserver**.

8. Now that the server is running, you can restore the security database with the ones you backed up before installing InterBase 6:

```
# gbak -r /tmp/isc4.gbak jupiter:/usr/interbase/isc4.gdb
```

9. To test your existing databases with InterBase 6, use **gbak** to restore the backup files you created before upgrading.

---

## Installing on Linux

For both the SuperServer and Classic versions of InterBase, default installation directories on Linux are as follows:

/opt/interbase	executables
/usr/lib	libraries
/usr/include	header files

---

InterBase SuperServer for Linux requires **glibc (libc6)** Version 2.1. InterBase does not work with the older Version 2.0 library.

### ► *Installing with Red Hat Package manager (RPM)*

1. Log into your database server as root.
2. Uninstall any previous versions of InterBase.
3. Install InterBase 6 using the RPM utility, for example:

```
rpm -i interbase-6.0-1.i386.rpm
```

4. Check that **/etc/hosts** contains the following line:

```
127.0.0.1 localhost
```

5. Check that **/etc/hosts.equiv** exists on the system and contains ‘localhost’. If the file already exists but does not contain ‘localhost’, edit the file to add it. If **/etc/hosts.equiv** does not exist, create it as follows:

```
echo localhost > /etc/hosts.equiv
```

**IMPORTANT** If **/etc/hosts.equiv** already exists on the system, then the command above will *overwrite* your current file, and all the information that was in it will be lost.

6. To test SuperServer installations do the following:

- Make sure that a search path (environment variable PATH) contains **/opt/interbase/bin**.
- Start the ibserver in **/opt/interbase/bin** with:

```
ibmgr -start -forever -user SYSDBA -password masterkey
```

- Start the **isql** utility and connect to the example database:

```
isql localhost:/opt/interbase/examples/employee.gdb
```

7. To test Classic installations do the following:

- Make sure that a search path (environment variable PATH) contains **/opt/interbase/bin**.
- Start the ibserver in **/opt/interbase/bin** with:

```
ibmgr -start -forever -user SYSDBA -password masterkey
```

**Note** The Classic architecture permits application processes to perform I/O on database files directly, whereas SuperServer requires applications to request the **ibserver** I/O operations by proxy, using a network method. The local access method is only usable by applications that run on the same host as the database. Thus we need to test both local and remote access for Classic servers.

- To test local access, start **isql** locally and connect to the example database:

```
isql /opt/interbase/examples/employee.gdb
```

- To test remote access, start the **isql** utility and connect to the example database:

```
isql localhost:/opt/interbase/examples/employee.gdb
```

### ► *Installing from a tar file*

1. Log into your database server as root.
2. Uninstall any previous versions of InterBase.
3. Extract the install script and the InterBase kit using the tar utility, for example:

```
tar xvf InterBaseSS_LI-B6.0.tar
```

4. Run the install script, for example:

```
./install /tmp/InterBase_LI-V6.0.tgz
```

5. Check that **/etc/hosts** contains the following line:

```
127.0.0.1 localhost
```

6. Check that **/etc/hosts.equiv** exists on the system and contains 'localhost'. If the file already exists but does not contain 'localhost', edit the file to add it. If **/etc/hosts.equiv** does not exist, create it as follows:

```
echo localhost > /etc/hosts.equiv
```

**IMPORTANT** If **/etc/hosts.equiv** already exists on the system, then the command above will *overwrite* your current file, and all the information that was in it will be lost.

7. To test SuperServer installations do the following:

- Make sure that a search path (environment variable PATH) contains **/opt/interbase/bin**.
- Start the ibserver in **/opt/interbase/bin** with:

```
ibmgr -start -forever -user SYSDBA -password masterkey
```

- Start the **isql** utility and connect to the example database:

```
isql localhost:/opt/interbase/examples/employee.gdb
```

8. To test Classic installations do the following:

- Make sure that a search path (environment variable PATH) contains **/opt/interbase/bin**.
- Start the ibserver in **/opt/interbase/bin** with:

```
ibmgr -start -forever -user SYSDBA -password masterkey
```

**Note** The Classic architecture permits application processes to perform I/O on database files directly, whereas SuperServer requires applications to request the **ibserver** I/O operations by proxy, using a network method. The local access method is only usable by applications that run on the same host as the database. Thus we need to test both local and remote access for Classic servers.

- To test local access, start **isql** locally and connect to the example database:

```
isql /opt/interbase/examples/employee.gdb
```

- To test remote access, start the **isql** utility and connect to the example database:

```
isql localhost:/opt/interbase/examples/employee.gdb
```

**Note** Change all instances of **/usr/interbase** to **/opt/interbase**.

---

## Running multiple versions on UNIX

We support both SuperServer and Classic architectures on Linux, and you can switch between versions of the same architecture as well as different architectures on the same host. To start a specific version, start the server in the directory to which the symbolic link points. To start the server daemon, follow the steps in “UNIX daemon” in the *Operations Guide*.

---

## Uninstalling InterBase on UNIX

On Solaris, use **pkgrm** to remove the InterBase 6 package:

```
# pkgrm interbase
```

On Linux, if InterBase was installed using RPM, use RPM to uninstall InterBase, for example:

```
rpm -e NameofInterBasePackage
```

On Linux, if InterBase was installed using a tar file, follow these steps to uninstall InterBase.

### ■ SuperServer:

1. If **ibserver** is running, use the following command to shut down the server:

```
ibmgr -shut -pass password
```

2. Copy the **ibinstall.log** file from the InterBase install directory (**/opt/interbase**) to **/tmp**.
3. Delete the InterBase install directory.
4. Remove all the InterBase files from the **/usr/lib** and **/usr/include** directories; see **/tmp/ibinstall.log** for the list of files.
5. Remove the InterBase entry from **/etc/services**; see **/tmp/ibinstall.log** for the InterBase entry.

### ■ Classic:

1. If **gds\_lock\_mgr** is running, use the following command to shut down the lock manager:

```
gds_drop -s
```

2. Copy the **ibinstall.log** file from the InterBase install directory (**/opt/interbase**) to **/tmp**.
3. Delete the InterBase install directory.



4. Remove all the InterBase files from the `/usr/lib`, `/usr/local/sbin`, and `/usr/include` directories; see `/tmp/ibinstall.log` for the list of files.
5. Remove the InterBase entry from `/etc/inetd.conf`; see `/tmp/ibinstall.log` for the InterBase entry.
6. Find the `inetd` process id in `/var/run/inetd.pid` and send the signal `SIGHUP` to the `inetd` process to tell it to re-read the configuration file and not listen on port 3050 (`gds_db`) any more.
7. Remove the InterBase entry from `/etc/services`; see `/tmp/ibinstall.log` for the InterBase entry.

---

## Installing and accessing InterBase documentation

You can install the InterBase documentation in PDF form on your hard drive; this requires about 24 MB of disk space. Or, you can install shortcuts to the documentation, in which case you must insert the InterBase CD-ROM each time you want to refer to the documentation.

You need Adobe Acrobat Reader With Search to view and search the documentation. The InterBase CD-ROM includes Acrobat Reader With Search (English version) of UNIX and Windows platforms. The files are in subdirectories of the `/Adobe` directory on the InterBase CD-ROM.





To access all of the books from a single document, open [2AllBooks.pdf](#). It contains links to the rest of the document set.


The InterBase document set consists of the *API Guide*, the *Data Definition Guide*, the *Developer's Guide*, the *Embedded SQL Guide* (formerly the *Programmer's Guide*), *Getting Started*, the *Language Reference*, the *Operations Guide*, and *Release Notes*.

---

## Full-text searching

The document set has been indexed for full-text searching. If you are viewing the documents using Acrobat Reader With Search, you can enter a query and receive a list of hits from all five books in the InterBase document set.

To use full-text searching, click the  button and search for a word or phrase. Acrobat Reader returns a list of books that contain the phrase. Choose the book you want to start looking in to display the first instance. You then use the  and  buttons to step forward and back through instances of your search target. Reader moves from one book to the next. To go to a different book at will, click the  button to display the “found” list.

Note that full-text searching is not the same as Find () , which searches only the current document.

*Tip* On UNIX and Linux, always open documents using an absolute pathname to the PDF file, to make Acrobat Reader With Search associate the index with the PDF document correctly.

---

## Links

The PDF documentation set contains many hypertext links that take you to referenced points in the document with a single click. In addition, the Table of Contents and Index entries are hypertext links and therefore are clickable. Throughout the document set, clickable links appear **bold and green**.

---

## Installing Acrobat Reader With Search

### On Windows

You can install Acrobat Reader 3.01 With Search by choosing Install Adobe Acrobat Reader 3.0 from the InterBase Launcher. You can also install it directly by running **setup.exe** from the **/Adobe** directory of the InterBase CD-ROM.

### On UNIX

The InterBase CD-ROM includes Acrobat Reader With Search for both the UNIX platform and for Windows. The files are in subdirectories of the **/Adobe** directory on the InterBase CD-ROM. Read **instguid.txt** in the **/Adobe/UNIX platform** directory for UNIX installation instructions. To install on a Windows platform, run **setup.exe** in the **/Adobe/Windows** directory.

### From the Adobe website

If you don't have the InterBase CD-ROM handy, you can get Acrobat Reader for free from the Adobe website. It's at <http://www.adobe.com/products/acrobat>. Check the box next to "Include option for searching PDF files..." to download Acrobat Reader With Search, not the plain Acrobat Reader.

# Migrating to InterBase 6

This chapter describes how to plan and execute a smooth migration from earlier versions of InterBase to InterBase 6. Topics in this chapter include:

- Migration issues: on-disk structure and dialects
- Migration paths
- Delphi, C++ Builder and InterBase 6: BDE and IBX

**Note** See the *Release Notes* for a detailed introduction to new InterBase features.

---

## Migration process

These are the steps you must take to migrate servers, databases, and clients. Each is discussed in detail in later sections:

- Server and database migration
  1. Backup all databases to be migrated
  2. Install the InterBase 6 server
  3. Restore databases to be migrated; at this point, you have dialect 1 databases
  4. Validate migrated databases
  5. Migrate databases to SQL dialect 3 (see pages 43 to 51)

- Client migration
  1. Identify the clients that must be upgraded to InterBase 6
  2. Identify areas in your application which may need upgrading
  3. Install the InterBase 6 client to each machine that requires it
  4. Upgrade SQL applications to SQL dialect 3

---

## Migration Issues

Before migrating your databases, you need to learn about InterBase 6 SQL dialects and understand their effect on servers, clients, and the use of new InterBase 6 features.

---

### InterBase 6 SQL dialects

InterBase recognizes different client and database dialects to allow users more mobility in how their legacy databases are used, accessed, and updated. In InterBase 6, each client and database has an *SQL dialect*: an indicator that instructs an InterBase 6 server how to interpret *transition features*: those features whose meanings have changed between InterBase versions. The following transition features have different meanings based on the dialect used by the client applications:

- Double quote (“): changed from a synonym for the single quote (‘) to the delimiter for an object name
- DECIMAL and NUMERIC datatypes with precision greater than 9: now stored as INT64 datatypes instead of DOUBLE PRECISION
- DATE, TIME, and TIMESTAMP datatypes: DATE has changed from a 64-bit quantity containing both date and time information to a 32-bit quantity containing only date information. TIME is a 32-bit quantity containing only time information, while TIMESTAMP is a 64-bit quantity containing both date and time information (the same as DATE in pre-Version 6 SQL).

---

### Servers, databases, and ODS 10

InterBase 6 databases are stored in the ODS 10 data format. This format is backward-compatible with older formats.

- If you upgrade a server to InterBase 6, you *have the option to* migrate the databases that it accesses to InterBase 6 as well.

- If you do not upgrade a server, you do not need to migrate the databases that it accesses. InterBase 5 and older servers can access InterBase 6 databases.

---

## Clients and databases

Clients and databases each have dialects. Servers do not themselves have a dialect, but they interpret data structures and client requests based on the dialect of each. Applications using an older version of the InterBase client work with the InterBase 6 server and its databases with some restrictions:

- Version 5 clients cannot access dialect 3 columns that are stored as INT64, TIME, or DATE. (DECIMAL and NUMERIC columns with precision greater than 9 are stored as INT64.)
- Version 5 clients cannot display new datatypes in metadata using the SHOW command, or any equivalent.
- Version 5 clients interpret the DATE datatype as TIMESTAMP, since that was the definition of DATE prior to InterBase 6.
- Version 5 clients cannot access any object named with a delimited identifiers.
- Clients that use the Borland Database Engine (BDE) to access an InterBase 6.0 server are not able to access any of the new field type regardless of the version of the InterBase client installed.

---

## Keywords used as identifiers

Version 5 clients have one advantage over version 6 clients: If you migrate an older database that uses some of the new version 6 keywords as identifiers to version 6 dialect 1, these older version 5 clients can still access those keyword objects. Version 6 dialect 1 cannot do so. Dialect 3 clients can access these keyword objects if they are delimited in double quotes.

If version 5 clients use any of the new InterBase 6 keywords as object names, the InterBase 6 server permits this without error because it recognizes that these clients were created at a time when these were not keywords.

*Example* For example, the following statement uses the new keyword word TIME:

```
SELECT TIME FROM atable;
```

This statement, when executed via a pre-InterBase 6 client returns the information as it did in previous versions. If this same query is issued using a version 6 client, an error is returned since TIME is now a reserved word. See page 27 for a list of new keywords.

## Understanding SQL dialects

Below are explanations of server and client behavior with SQL dialects 1, 2, and 3.

### ► *Dialect 1 clients and databases*

In dialect 1, the InterBase 6 server interprets transition features just like an InterBase 5 server:

- Double quoted text is interpreted as a string literal. Delimited identifiers are not available.
- The DATE datatype contains both time and date information and is interpreted as TIMESTAMP; the name has changed but the meaning has not. Dialect 1 clients expect the entire timestamp to be returned. In dialect 1, DATE and TIMESTAMP are identical.
- The TIME datatype is not available.
- *Dialect 1 databases store* DECIMAL and NUMERIC datatypes with precision greater than 9 as DOUBLE PRECISION, not INT64. *In clients:* The dialect 1 client expects information stored in these datatypes to be returned as double precision; such clients cannot create database fields to hold 64-bit integers.

An InterBase 6 server recognizes all the other InterBase 6 features in dialect 1 clients and databases.

### ► *Dialect 2 clients*

Dialect 2 is available only on the client side. It is intended for assessing possible problems in legacy metadata that is being migrated to dialect 3. To determine where the problem spots are when you migrate a database from dialect 1 to dialect 3, you extract the metadata from the database, set **isql** to dialect 2, and then run that metadata file through **isql**. **isql** issues warning whenever it encounters double quotes, DATE datatypes, or large exact numerics to alert you to places where you might need to change the metadata in order to make a successful migration to dialect 3.

To detect problem areas in the metadata of a database that you are migrating, extract the metadata and run it through a dialect 2 client, which will report all instances of transition features. For example:

```
isql -i v5metadata.sql
```

Do not assign dialect 2 to databases.

### ► *Dialect 3 clients and databases*

In dialect 3, the InterBase server interprets transition features as InterBase 6 SQL 92-compliant:

- Double quoted strings are treated as delimited identifiers.

- *Dialect 3* DATE datatype fields contain only date information. Dialect 3 clients expect only date information from a field of datatype DATE.
- The TIME datatype is available, and stores only time information.
- *Dialect 3 databases store* DECIMAL and NUMERIC datatypes with precision greater than 9 as INT64 *if and only if they are in columns that were created in dialect 3*. Dialect 3 clients expect DECIMAL and NUMERIC datatypes with precision greater than 9 to be returned as INT64. (To learn how to migrate older data to INT64 storage, see **“Migrating databases to dialect 3” on page 43** and **“Migrating NUMERIC and DECIMAL datatypes” on page 48**.)

## Setting SQL dialects

You can set the SQL dialect for a server or client in a variety of ways. For example, the new IBConsole user interface has menu options to specify SQL dialect. See the *Operations Guide* for a complete explanation of IBConsole use. This section explores the command-line methods of setting dialect.

### Setting isql client dialect

To use **isql** to create a database in a particular dialect, first set **isql** to the desired dialect and then create the database. You can set **isql** dialect in the following ways:

- On the command line, start **isql** with option **-sql\_dialect *n***, where *n* is 1, 2, or 3.

```
isql -sql_dialect n
```

- Within an **isql** session or in an SQL script, you can issue this statement:

```
SET SQL DIALECT n
```

The following table shows the precedence for setting **isql** dialect:

Ranking	How dialect is set
Lowest	Dialect of an attached Version 6 database
Next lowest	Dialect specified on the command line: <pre>isql -sql_dialect n</pre>
Next highest	Dialect specified during the session: <pre>SET SQL DIALECT n;</pre>
Highest	Dialect of an attached Version 5 database (=1)

TABLE 3.1 isql dialect precedence

In InterBase 6, **isql** has the following behavior with respect to dialects:

- If you start **isql** and attach to a database without specifying a dialect, **isql** takes on the dialect of the database.
- If you specify a dialect on the command line when you invoke **isql**, it retains that dialect after connection unless explicitly changed.
- When you change the dialect during a session using `SET SQL DIALECT n`, **isql** continues to operate in that dialect until explicitly changed.
- When you create a database using **isql**, the database is created with the dialect of the **isql** client; for example, if **isql** has been set to dialect 1, when you create a database, it is a dialect 1 database.
- If you create a database without first specifying a dialect for the **isql** client or attaching to a database, **isql** creates the database in dialect 3.

The statements above are true whether you are running **isql** as a command-line utility or are accessing it through IBConsole, InterBase's new interface.

**IMPORTANT** Any InterBase 6 **isql** client that attaches to a version 5 database resets to dialect 1.

---

## Setting the gpre client dialect

In InterBase 6, by default **gpre** takes on the dialect of the database to which it is connected. This enables **gpre** to parse pre-Version 6 source files without moderation.

You can set **gpre** to operate as a client in a different dialect these ways:

- Start **gpre** with option `-sql_dialect n`. For example, this command sets **gpre** to dialect 3:

```
gpre -sql_dialect 3
```

- Specify dialect within the source, for example:

```
EXEC SQL
    SET SQL DIALECT n
```

The precedence of dialect specification is as follows:

Lowest	Dialect of an attached database
Middle	Command line specification: <code>gpre -sql_dialect <i>n</i></code>
Highest	Dialect explicitly specified within the source, for example <code>EXEC SQL</code> <code>SET SQL DIALECT <i>n</i></code>



---

## Setting the gfix client dialect

The command-line option `-sql_dialect n`, where *n* is 1 or 3, sets the dialect of an ODS 10 version database. For example, the following statement sets **mydb.gdb** to dialect 3:

```
gfix -sql_dialect 3 mydb.gdb
```

---

## New features

Many of the new InterBase 6 features operate without reference to dialect. Other features are dialect-specific. These features are listed below and described in more detail in the *Release Notes and other InterBase manuals*.

---

### Features available in all dialects

The following new features are available in both dialects of InterBase 6:

- IBConsole, InterBase's new graphical interface

IBConsole, InterBase's new graphical user interface, combines the functionality of Server Manager and InterBase Windows ISQL. You now create and maintain databases, configure and maintain servers, and execute interactive SQL from one integrated interface.

- Read-only databases

You can make InterBase 6 databases be read-only. This permits distribution on read-only media such as CDROMs and reduces the chance of accidental or malicious changes to databases.

- Altering column definitions

The ALTER COLUMN clause of the ALTER TABLE statement can change a column's name, datatype, or position.

- Altering domain definitions

ALTER DOMAIN now includes the ability to change the name or datatype of a domain definition.

- The EXTRACT() function

The new EXTRACT() function extracts information from the new DATE, TIMESTAMP, and TIME datatypes. In dialect 1, you can use it to extract information from the TIMESTAMP datatype. **Note** "DATE" is new in the sense that it has a different meaning in dialect 3 databases than it did previously.

- SQL warnings

The InterBase API function set now returns warnings and informational messages along with error messages in the status vector.

- The Services API, Install API, and Licensing API

InterBase now provides three new function libraries. The Services API, which is part of the InterBase client library, provides functions for database maintenance tasks, software activation, requesting information about the configuration of databases and server, and working with user entries in the security database.

- New **gbak** functionality

In InterBase 6, **gbak** can

- Back up to multiple files and restore to multiple files
- Use the **-service** switch to perform server-side backups and restores
- Set databases to read-only mode when restoring

- InterBase Express™ (IBX)

IBX provides native Delphi components for InterBase data access, services, and installation. Borland C++ Builder also can access IBX components.

---

## Features available only in dialect 3 databases

The following features are available only in dialect 3 clients and databases because they conflict with dialect 1 usage.

- Delimited identifiers

Identifiers can now be keywords, contain spaces, be case sensitive, and contain non-ASCII characters. Such identifiers must be delimited by double quotes. String constants must be delimited by single quotes.

- INT64 data storage

In dialect 3 databases, data stored in DECIMAL and NUMERIC columns is stored as INT64 when the precision is greater than 9. This is true only for columns that are *created* in dialect 3. These same datatypes are stored as DOUBLE PRECISION in dialect 1 and in all older InterBase versions. This change in storage also requires different arithmetic algorithms.

- DATE and TIME datatypes

In dialect 3, the DATE datatype holds only date information. This is a change from earlier InterBase versions in which it stored the whole timestamp.

Dialect 3 allows the use of the TIME datatype, which hold only the time portion of the timestamp.

---

## New keywords

InterBase 6 introduces the following new keywords:

COLUMN	SECOND
CURRENT_DATE	TIME
CURRENT_TIME	TIMESTAMP
CURRENT_TIMESTAMP	TYPE
DAY	WEEKDAY
EXTRACT	YEAR
HOURL	YEARDAY
MINUTE	
MONTH	

These keywords are reserved words in all version 6 dialects.

- You cannot create objects in a version 6 dialect 1 database that have any of these keywords as object names (identifiers).
- You can migrate a version 5 database that contains these keywords used as identifiers to version 6 dialect 1 without changing the object names: a column could be named “YEAR”, for instance.
  - Version 5 clients can access these keyword identifiers without error.
  - Version 6 clients *cannot* access keywords that are used as identifiers. In a dialect 1 database, you must change the names so that they are not keywords.
  - If you migrate directly to dialect 3, you can retain the names, but you must delimit them with double quotes. To retain accessibility for older clients, put the names in all upper case. Delimited identifiers are case sensitive.
- Although TIME is a reserved word in version 6 dialect 1, you cannot use it as a datatype because such databases guarantee datatype compatibility with version 5 clients.
- In dialect 3 databases and clients, any reserved word can be used as an identifier as long as it is delimited with double quotes.

---

## Delimited identifiers

To increase compliance with the SQL 92 standard, InterBase 6 introduces *delimited identifiers*. An identifier is the name of any database object; for instance a table, a column, or a trigger. A delimited identifier is an identifier that is enclosed in double quotes. Because the quotes delimit the boundaries of the name, the possibilities for object names are greatly expanded from previous versions of InterBase. Object names can now:

- mimic keywords
- include spaces (except trailing spaces)
- be case-sensitive

### ► *Double-quotes use*

Up to and including version 5, InterBase allowed the use of either single or double quotes around string constants. The concept of delimited identifiers did not exist. InterBase 6 operates under these rules for double quotes:

- In all versions of InterBase, anything delimited by single quotes is treated as a string constant.
- In version 5 and older of InterBase, string constants were delimited by either double or single quotes. Since there was no concept of delimited identifiers, double quotes were always interpreted as string constants.
- Version 6 dialect 1 is a transition mode that behaves like older versions of InterBase with respect to quote marks: it interprets strings within double quotes as string constants and does not permit delimited identifiers.
- Version 6 dialect 3 uses double quotes only for delimited identifiers. String constants must be delimited by single quotes, never double.
- When version 6 servers detect that the client is dialect 1, they permit client DML statements to contain double quotes and they correctly handle these as string constants. However, they do not permit double quotes in client DDL statements because that metadata would not be allowed in dialect 3. Version 6 servers all insist that string constants be delimited with single quotes when clients create new metadata.

## DATE, TIME, and TIMESTAMP datatypes

InterBase 6 dialect 3 replaces the old InterBase DATE datatype, which contains both date and time information, with SQL-92 standard TIMESTAMP, DATE, and TIME datatypes. The primary migration problem exists in the source code of application programs that use the InterBase 5 DATE datatype. In InterBase 6, the DATE keyword represents a date-only datatype, while a Version 5 DATE represents a date-and-time datatype.

Columns and domains that are defined as DATE datatype in InterBase 5 appear as TIMESTAMP columns when the database is restored in InterBase 6. However, a TIMESTAMP datatype has four decimal points of precision, while a Version 5 DATE datatype has only two decimal points of precision.

If you migrate your database to dialect 3 and you require only date or only time information from a TIMESTAMP column, you can use ALTER COLUMN to change the datatype to DATE or TIME. These columns each take only four bytes, whereas TIMESTAMP and the InterBase 5 DATE columns each take eight bytes. If your TIMESTAMP column holds both date and time information, you cannot change it to an InterBase 6 DATE or TIME column using ALTER COLUMN, because ALTER COLUMN does not permit data loss. Dialect use also enforces certain rules:

- In dialect 1, only TIMESTAMP is available. TIMESTAMP is the equivalent of the DATE datatype in previous versions. When you back up an older database and restore it in version 6, all the DATE columns and domains are automatically restored as TIMESTAMP. DATE and TIMESTAMP datatypes are both available and both mean the same thing in dialect 1.
- In dialect 3, TIMESTAMP functions as in dialect 1, but two additional datatypes are available: DATE and TIME. These datatypes function as their names suggest: DATE holds only date information and TIME holds only time.
- In dialect 3, DATE and TIME columns require only four bytes of storage, while TIMESTAMP columns require eight bytes.

The following example shows the differences between dialect 1 and dialect 3 clients when date information is involved.

*Example*    `CREATE TABLE table1 (fld1 DATE, fld2 TIME);`  
               `INSERT INTO table1 VALUES (CURRENT_DATE, CURRENT_TIME);`

### Using dialect 1 clients

```
SELECT * FROM table1;

Statement failed, SQLCODE = -804
Dynamic SQL Error
-SQL error code = -804
```

-Data type unknown  
 -Client SQL dialect 1 does not support reference to TIME datatype

```
SELECT fld1 FROM table1;
```

Statement failed, SQLCODE = -206

Dynamic SQL Error

-SQL error code = -206

-Column unknown

-FLD1

-Client SQL dialect 1 does not support reference to DATE datatype

### Using dialect 3 clients

```
SELECT * FROM table1;
```

```
      FLD1          FLD2
=====
1999-06-25  11:32:30.0000
```

```
SELECT fld1 FROM table1;
```

```
      FLD1
=====
1999-06-25
```

*Example* CREATE TABLE table1 (fld1 TIMESTAMP);  
 INSERT INTO table1 (fld1) VALUES (CURRENT\_TIMESTAMP);  
 SELECT \* FROM table1;

### In dialect 1:

```
      FLD1
=====
25-JUN-1999
```

### In dialect 3:

```
      FLD1
=====
1999-06-25 10:24:35.0000
```

*Example* SELECT CAST (fld1 AS CHAR(5)) FROM table1;

### In dialect 1:

```
=====
```

25-JU

**In dialect 3:**

```
Statement failed, SQLCODE = -802
arithmetic exception, numeric overflow, or string truncation
```

**► *Converting TIMESTAMP columns to DATE or TIME***

Once you have migrated a database to dialect 3, any columns that previously had the DATE datatype will have the TIMESTAMP datatype. If you want to store that data in a DATE or TIME column, follow these steps:

1. Use ALTER TABLE to create a new column of the desired type.
2. Insert the values from the original column into the new column:

```
UPDATE tablename SET new_field = CAST (old_field AS new_field);
```

3. Use ALTER TABLE to drop the original column.
4. Use ALTER TABLE ... ALTER COLUMN to rename the new column.

**► *Casting date/time datatypes***

InterBase 6 dialect 3 no longer allows the use of the CAST operator to remove the data portion of a timestamp by casting the timestamp value to a character value. When you cast a TIMESTAMP to a CHAR or VARCHAR in dialect 3, the destination type must be at least 24 characters in length or InterBase will report a string overflow exception. This is required by the SQL3 standard.

You can use the CAST() function in SELECT statements to translate between date/time datatypes and various character-based datatypes. The character datatype must be at least 24 characters in length. You can, however, cast a TIMESTAMP to a DATE and then cast the DATE to a CHAR of less than 24 characters. For example:

```
SELECT CAST (CAST (timestamp_col AS DATE) AS CHAR(10)) FROM table1;
```

It is not possible to cast a date/time datatype to or from BLOB, SMALLINT, INTEGER, FLOAT, DOUBLE PRECISION, NUMERIC, or DECIMAL datatypes.

For more information, refer to “Using CAST() to convert dates and times” in the *Embedded SQL Guide*.

Table 3.2 outlines the results of casting *to* date/time datatypes:

Cast From	To		
	TIMESTAMP	DATE	TIME
VARCHAR( <i>n</i> ) CHARACTER( <i>n</i> ) CSTRING( <i>n</i> )	String must be in format YYYY-MM-DD HH:MM:SS.thousands	See below.	String must be in format HH:MM:SS.thousands
TIMESTAMP	Always succeeds	Date portion of timestamp	Time portion of timestamp
DATE	Always succeeds; time portion of timestamp set to 0:0:0.0000	Always succeeds	Error
TIME	Always succeeds; date portion of timestamp set to current date	Error	Always succeeds

TABLE 3.2 Results of casting to date/time datatypes

**Casting DATE to string** results in YYYY-MM-DD where “MM” is a two-digit month. If the result does not fit in the string variable a string truncation exception is raised. In earlier versions, this case results in DD-Mon-YYYY HH:mm:SS.hundreds where “Mon” was a 3-letter English month abbreviation. Inability to fit in the string variable resulted in a silent truncation.

**Casting a string to a date** now permits strings of the form:

'yyyy-mm-dd'      'yyyy/mm/dd'      'yyyy mm dd'  
'yyyy:mm:dd'      'yyyy.mm.dd'

In all of the forms above, you can substitute a month name or 3-letter abbreviation in English for the 2-digit numeric month. However, the order must always be 4-digit year, then month, then day.

In previous versions of InterBase, you could enter date strings in a number of forms, including ones that had only two digits for the year. Those forms are still available in InterBase 6. If you enter a date with only two digits for the year, InterBase uses its "sliding window" algorithm to assign a century to the years.

The following forms were available in earlier versions of InterBase and are still permitted in InterBase 6:

'mm-dd-yy'      'mm-dd-yyyy'      'mm/dd/yy'      'mm/dd/yyyy'  
'mm dd yy'      'mm dd yyyy'      'mm:dd:yy'      'mm:dd:yyyy'  
'dd.mm.yy'      'dd.mm.yyyy'



If you write out the month name in English or use a 3-character English abbreviation, you can enter either the month or the day first. In the following examples, “xxx” stands for either a whole month name or a three-letter abbreviation. All of the following forms are acceptable:

'dd-xxx-yy'	'dd-xxx-yyyy'	'xxx-dd-yy'	'xxx-dd-yyyy'
'dd xxx yy'	'dd xxx yyyy'	'xxx dd yy'	'xxx dd yyyy'
'dd:xxx:yy'	'dd:xxx:yyyy'	'xxx:dd:yy'	'xxx:dd:yyyy'

For example, the following INSERT statements all insert the date “January 22, 1943”:

```
INSERT INTO t1 VALUES ('1943-01-22');
INSERT INTO t1 VALUES ('01/22/1943');
INSERT INTO t1 VALUES ('22.01.1943');
INSERT INTO t1 VALUES ('jan 22 1943');
```

The following statement would enter the date “January 22, 2043”:

```
INSERT INTO t1 VALUES ('01/22/43');
```

Table 3.3 outlines the results of casting *from* date/time datatypes:

Cast From	To VARCHAR( <i>n</i> ), CHARACTER ( <i>n</i> ), or CSTRING( <i>n</i> )
TIMESTAMP	Succeeds if <i>n</i> is 24 or more. Resulting string is in format YYYY-MM-DD HH:MM:SS.thousands.
DATE	Succeeds if <i>n</i> is 10 or more. Resulting string is in the format YYYY-MM-DD.
TIME	Succeeds if <i>n</i> is 13 or more. Resulting string is the format HH:MM:SS.thousands.

TABLE 3.3 Results of casting to date/time datatypes

► *Adding and subtracting datetime datatypes*

The following table shows the result of adding and subtracting DATE, TIME, TIMESTAMP, and numeric values. “Numeric value” refers to any value that can be cast as an exact numeric value by the database engine (for example, INTEGER, DECIMAL, or NUMERIC).

Operand1	Operator	Operand2	Result
DATE	+	DATE	Error
DATE	+	TIME	TIMESTAMP (concatenation)
DATE	+	TIMESTAMP	Error
DATE	+	Numeric value	DATE + number of days: fractional part ignored
TIME	+	DATE	TIMESTAMP (concatenation)
TIME	+	TIME	Error
TIME	+	TIMESTAMP	Error
TIME	+	Numeric value	TIME + number of seconds: 24-hour modulo arithmetic
TIMESTAMP	+	DATE	Error
TIMESTAMP	+	TIME	Error
TIMESTAMP	+	TIMESTAMP	Error
TIMESTAMP	+	Numeric value	TIMESTAMP: DATE + number of days; TIME + fraction of day converted to seconds
DATE	–	DATE	DECIMAL(9,0) representing the number of days
DATE	–	TIME	Error
DATE	–	TIMESTAMP	Error
DATE	–	Numeric value	DATE – number of days: fractional part ignored
TIME	–	DATE	Error
TIME	–	TIME	DECIMAL(9,4) representing number of seconds
TIME	–	TIMESTAMP	Error

TABLE 3.4 Adding and subtracting date/time datatypes

Operand1	Operator	Operand2	Result
TIME	–	Numeric value	TIME – number of seconds: 24-hour modulo arithmetic
TIMESTAMP	–	DATE	Error
TIMESTAMP	–	TIME	Error
TIMESTAMP	–	TIMESTAMP	DECIMAL(18,9) representing days and fraction of day
TIMESTAMP	–	Numeric value	TIMESTAMP: DATE – number of days; TIME – fraction of day converted to seconds

TABLE 3.4 Adding and subtracting date/time datatypes

**Note** Numeric value + DATE, TIME, or TIMESTAMP is symmetric to DATE, TIME, or TIMESTAMP + numeric value.

► *Using date/time datatypes with aggregate functions*

You can use the date/time datatypes with the MIN(), MAX(), COUNT() functions, the DISTINCT argument to those functions, and the GROUP BY argument to the SELECT() function. An attempt to use SUM() or AVG() with date/time datatypes returns an error.

► *Default clauses*

CURRENT\_DATE, CURRENT\_TIME, and CURRENT\_TIMESTAMP can be specified as the default clause for a domain or column definition.

► *Extracting date and time information*

The EXTRACT() function extracts date and time information from databases. In dialect 3, the EXTRACT operator allows you to return different parts of a TIMESTAMP value. The EXTRACT operator makes no distinction between dialects when formatting or returning the information. EXTRACT() has the following syntax:

```
EXTRACT (part FROM value)
```

The value passed to the EXTRACT() expression must be DATE, TIME, or TIMESTAMP. Extracting a part that doesn't exist in a datatype results in an error. For example:

```
EXTRACT (TIME FROM aTime)
```

A statement such as EXTRACT (YEAR from aTime) would fail.

The datatype of `EXTRACT()` expressions depends on the specific part being extracted:

<b>Extract</b>	<b>Resulting datatype</b>	<b>Representing</b>
YEAR	SMALLINT	Year, range 0-5400
MONTH	SMALLINT	Month, range 1-12
DAY	SMALLINT	Day, range 1-31
HOURL	SMALLINT	Hour, range 1-23
MINUTE	SMALLINT	Minute, range 1-59
SECOND	DECIMAL(6,4)	Second, range 0-59.9999
WEEKDAY	SMALLINT	Day of the week, range 0-6 (0 = Sunday, 1 = Monday, and so on)
YEARDAY	SMALLINT	Day of the year, range 1-366

TABLE 3.5 Extracting date and time information

```

SELECT EXTRACT (YEAR FROM timestamp_fld) FROM table_name;
=====
1999

SELECT EXTRACT (YEAR FROM timestamp_fld) FROM table_name;
=====
1999

SELECT EXTRACT (MONTH FROM timestamp_fld) FROM table_name;
=====
6

SELECT EXTRACT (DAY FROM timestamp_fld) FROM table_name;
=====
25

SELECT EXTRACT (MINUTE FROM timestamp_fld) FROM table_name;
=====
24

SELECT EXTRACT (SECOND FROM timestamp_fld) FROM table_name;
=====
35.0000

SELECT EXTRACT (WEEKDAY FROM timestamp_fld) FROM table_name;

```

```

=====
5

SELECT EXTRACT (YEARDAY FROM timestamp_fld) FROM table_name;

=====
175

SELECT EXTRACT (MONTH FROM timestamp_fld) ||
'-' || EXTRACT (DAY FROM timestamp_fld) ||
'-' || EXTRACT (YEAR FROM timestamp_fld) FROM table_name;

=====
6-25-1999

```

---

## DECIMAL and NUMERIC datatypes

The following sections highlight some of the changes introduced by InterBase 6 when dealing with numeric values. They need to be considered carefully when migrating your database from dialect 1 to dialect 3. When considering these issues, keep in mind that in order to make use of the new functionality, the statements must be created with a client dialect setting of 3.

The most notable migration issues involve using the division operator and the AVG() function (which also implies division) with exact numeric operands. *Exact numeric* refers to any of the following data types: INTEGER, SMALLINT, DECIMAL, NUMERIC. NUMERIC and DECIMAL datatypes that have a precision greater than 9 are called “large exact numerics” in this discussion. Large exact numerics are stored as DOUBLE PRECISION in dialect 1 and as INT64 in columns created in dialect 3.

**IMPORTANT** When you migrate an exact numeric column to dialect 3 it is still stored as DOUBLE PRECISION. The migration does not change the way the data is stored because INT64 cannot store the whole range that DOUBLE PRECISION can store. There is potential data loss, so InterBase does not permit direct conversion. If you decide that you want your data stored as INT64, you must create a new column and copy the data. Only exact numeric columns that are *created* in dialect 3 are stored as INT64. The details of the process are provided in **“Migrating databases to dialect 3” on page 43**.

You might or might not want to change exact numeric columns to INT64 when you migrate to dialect 3. See **“Do you really need to migrate your NUMERIC and DECIMAL datatypes?”** on page 48 for a discussion of issues.

Dialect 3 features and changes include

- Support for 64 bit integers.

- **Overflow protection.** In dialect 1, if the product of two integers was bigger than 31 bits, the product was returned modulo  $2^{32}$ . In dialect 3, the true result is returned as a 64-bit integer. Further, if the product, sum, difference, or quotient of two exact numeric values is bigger than 63 bits, InterBase issues an arithmetic overflow error message and terminates the operation. (Previous versions sometimes returned the least-significant portion of the true result.). The stored procedure **bignum** below demonstrates this.

Operations involving division return an exact numeric if both operands are exact numerics in dialect 3. When the same operation is performed in dialect 1, the result is a DOUBLE PRECISION.

To obtain a DOUBLE PRECISION quotient of two exact numeric operands in dialect 3, explicitly cast one of the operands to DOUBLE PRECISION before performing the division:

```
CREATE TABLE table 1 (n1 INTEGER, n2 INTEGER);
INSERT INTO table 1 (n1, n2) VALUES (2, 3);
SELECT n1 / n2 FROM table1;
```

```
=====
0
```

Similarly, to obtain a double precision value when averaging an exact numeric column, you must cast the argument to double precision before the average is calculated:

```
SELECT AVG(CAST(int_col AS DOUBLE PRECISION)) FROM table1;
```

---

## Compiled objects

The behavior of a compiled object such as a stored procedure, trigger, check constraint, or default value depends on the dialect setting of the client at the time the object is compiled. Once compiled and validated by the server the object is stored as part of the database and its behavior is constant regardless of the dialect of the client that calls it.

*Example* Consider the following procedure:

```
CREATE PROCEDURE exact1 (a INTEGER, b INTEGER) RETURNS (c INTEGER)
AS BEGIN
    c = a / b;
    EXIT;
END;
```

### When created by a dialect 1 client:

EXECUTE PROCEDURE *exact 1* returns 1 when executed by either a dialect 1 or dialect 3 client.

**When created by a dialect 3 client:**

EXECUTE PROCEDURE *exact 1* returns 0 when executed by either a dialect 1 or dialect 3 client.

*Example* Consider the following procedure:

```
CREATE PROCEDURE bignum (a INTEGER, b INTEGER) RETURNS (c NUMERIC(18,0))
AS BEGIN
    c = a * b;
    EXIT;
END;
```

**When created by a dialect 1 client:**

EXECUTE PROCEDURE *bignum* (65535, 65535) returns -131071.0000 when executed by either a dialect 1 or dialect 3 client.

**When created by a dialect 3 client:**

EXECUTE PROCEDURE *bignum* (65535, 65535) returns \*ERROR\* can't access INT64 when executed by a dialect 1 client.

EXECUTE PROCEDURE *bignum* (65535, 65535) returns 4294836225 when executed by a dialect 3 client.

---

## Generators

InterBase 6 generators return a 64-bit value, and only wrap around after  $2^{64}$  invocations (assuming an increment of 1), rather than after  $2^{32}$  as in InterBase 5. Applications should use an *ISC\_INT64* variable to hold the value returned by a generator. A client using dialect 1 receives only the least significant 32 bits of the updated generator value, but the entire 64-bit value is incremented by the engine even when returning a 32-bit value to a client that uses dialect 1. If your database was using an INTEGER field for holding generator values, you need to recreate the field so that it can hold 64-bit integer values.

---

## Miscellaneous issues

- IN clauses have a limit of 1500 elements

*Resolution* If you have more than 1500 elements, place the values in a temporary table and use a SELECT subquery in place of the list elements.

- Arithmetic operations on character fields are no longer permitted in client dialect 3

*Resolution* Explicitly cast the information before performing arithmetic calculations.

- Using **isql** to select from a **TIMESTAMP** column displays all information when client dialect is 3.

*Resolution* In versions of InterBase prior to 6.0, the time portion of a timestamp displayed only if **SET TIME ON** was in effect. In 6.0 client dialect 3, the time portion of the timestamp always displays.

---

## Migrating servers and databases

You can migrate your servers and applications to InterBase 6 at different times. They are separate migrations. Bear the following issues in mind as you plan your migration:

- Older clients can still access databases that have been migrated to InterBase 6. You must be aware, however, that they cannot access new datatypes or data stored as **INT64**, and they always handle double quoted material as strings.
- InterBase strongly recommends that you establish a migration testbed to check your migration procedures before migrating production servers and databases. The testbed does not need to be on the same platform as the production clients and servers that you are migrating.

The migration path varies somewhat depending on whether you are replacing an existing server or installing a new server and moving old databases there. Upgrading an existing server costs less in money, but may cost more in time and effort. The server and all the databases you migrate with it are unavailable during the upgrade. If you have hardware available for a new InterBase 6 server, the migration can be done in parallel, without interrupting service more than very briefly. This option also offers an easier return path if problems arise with the migration.

---

### “In-place” server migration

This section describes the recommended steps for replacing an InterBase 5 server with an InterBase 6 server.

1. Shut down each database before backup to ensure that no transactions are in progress.
2. Back up all databases on the version 5 server. Include **isc4.gdb** if you want to preserve your configured user IDs.

As a precaution, you should validate your databases before backing up and then restore each database to ensure that the backup file is valid.



3. Shut down the version 5 server. If your current server is a Superserver, you are not required to uninstall the server if you intend to install over it, although uninstalling is always good practice. You cannot have multiple versions of InterBase on the same machine. If your current server is Classic, you *must* uninstall before installing InterBase 6.

4. Install the version 6 server.

**Note** The install does not overwrite **isc4.gdb** or **isc4.gbk**.

5. Start the new server.

- On Windows NT, go to Services in the Control Panel and start the InterBase Guardian.
- On Windows 2000, go to **Control Panel | Administrative Tools | Services** and start the InterBase Guardian.
- On Windows 98/ME, run the InterBase Guardian application.
- On UNIX/Linux platforms, issue the following command to start the InterBase Superserver as user “interbase”:

```
# echo "/usr/interbase/bin/ibmgr -start -forever" | su interbase
```

Note that InterBase can run only as user “root” or user “interbase” on UNIX.

6. To restore the list of valid users, follow these steps:

- a. Restore **isc4.gbk** to **isc4\_old.gdb**
- b. Shut down the server
- c. Copy **isc4\_old.gdb** over **isc4.gdb**
- d. Copy **isc4\_old.gbk** over **isc4.gbk**
- e. Restart the server

7. Delete each ODS 9 database file. Restore each database from its backup file. This process creates InterBase 6, ODS 10, dialect 1 databases.

8. Perform a full validation of each database.

After performing these steps, you have an InterBase 6 server and InterBase 6, dialect 1 databases. See **“About InterBase 6, dialect 1 databases” on page 43** to understand more about these databases. See **“Migrating databases to dialect 3” on page 43** for a description of how to migrate databases to dialect 3. See **“Migrating clients” on page 51** for an introduction to client migration.

## Migrating to a new server

This section describes the recommended steps for installing InterBase 6 as a new server and then migrating databases from a previous InterBase 5 installation. The process differs only slightly from an in-line install.

1. Back up all databases on the version 5 server. Include **isc4.gdb** if you want to preserve your configured user IDs. Shut down the databases before backup to ensure that no transactions are in progress.
2. Install the version 6 server.
3. Start the new version 6 server.
  - On Windows NT/2000, go to Services in the Control Panel and start the InterBase Guardian.
  - On Windows 98, run the InterBase Guardian application.
  - On UNIX/Linux platforms, issue the following command to start the InterBase Superserver as user “interbase”:

```
# echo "/usr/interbase/bin/ibmgr -start -forever" | su interbase
```

Note that InterBase can run only as user “root” or user “interbase” on UNIX.

4. Copy the database backup files to the new server and restore each database from its backup file. This process creates InterBase 6, ODS 10, dialect 1 databases.

Save your backup files until your migration to dialect 3 is complete.

5. To restore the list of valid users, follow these steps:
  - a. Restore **isc4.gbk** to **isc4\_old.gdb**
  - b. Shut down the server
  - c. Copy **isc4\_old.gdb** over **isc4.gdb**
  - d. Copy **isc4\_old.gbk** over **isc4.gbk**
  - e. Restart the server
6. Perform a full validation of each database on the new server.

After performing these steps, you have an InterBase 6 server and InterBase 6, dialect 1 databases. See **“About InterBase 6, dialect 1 databases” on page 43** to understand more about these databases. See **“Migrating databases to dialect 3” on page 43** for a description of how to migrate databases to dialect 3. See **“Migrating clients” on page 51** for an introduction to client migration.

---

## About InterBase 6, dialect 1 databases

When you back up a version 5 database and restore it in InterBase 6, what do you have?

- A version 5 client can access everything in the database with no further changes.
- If there are object names—column or table names, for instance—that include any of the 17 new keywords, you must change these names in order to access these objects with a version 6 dialect 1 client. The new ALTER COLUMN clause of ALTER TABLE makes it easy to implement column name changes.
  - Version 5 clients can still access the columns.
  - Dialect 3 clients can access these columns as long as they delimit them with double quotes.
- The 17 new keywords are reserved words. However, the new datatypes TIME and DATE are not available to use as datatypes. DATE columns have the old meaning—both date and time. The new meaning of DATE—date only—is available only in dialect 3.
- All columns that were previously DATE datatype are now TIMESTAMP datatype. TIMESTAMP contains exactly the information that DATE did in previous versions.
- Exact numeric columns—those that have a DECIMAL or NUMERIC datatype with precision greater than 9—are still stored as DOUBLE PRECISION datatypes. All arithmetic algorithms that worked before on these columns still work as before. It is not possible to store data as INT64 in dialect 1.

---

## Migrating databases to dialect 3

There are four major areas of concern when migrating a database from dialect 1 to dialect 3: double quotes, the DATE datatype, large exact numerics (for purposes of this discussion, NUMERIC and DECIMAL datatypes that have a precision greater than 9), and keywords.

The process varies somewhat depending on whether you can create an application to move data from your original database to an empty dialect 3 database. If you do not have access to such a utility, you need to perform an in-place migration of the original database.

---

### Overview

In either method, you begin by extracting the metadata from your database, examining it for problem areas, and fixing the problems.

- If you are performing an in-place migration, you copy corrected SQL statements from the metadata file into a new script file, modify them, and run the script against the original database. Then you set the database to dialect 3. There are some final steps to take in the dialect 3 database to store old data as INT64.
- If you have a utility for moving data from the old database to a newly created empty database, you use the modified metadata file to create a new dialect 3 database and use the utility to transfer data from the old database to the new.

In both cases, you must make changes to the new database to accommodate migrated columns that must be stored as INT64 and column constraints and defaults that originally contained double quotes.

The two methods are described below.

---

### Method one: in-place migration

1. If you have not migrated the database to version 6, dialect 1, do so first. Back up the database again.
2. Extract the metadata from the database using `isql -x`. If you are migrating legacy databases that contain GDML, see **“Migrating older databases” on page 50**.
3. Prepare an empty text file to use as a script file. As you fix data structures in the metadata files, you will copy them to this file to create a new script.

**Note** You could also proceed by removing unchanged SQL statements from the original metadata file, but this is more likely to result in problems from statements that were left in error. InterBase recommends creating a new script file that contains only the statements that need to be run against the original database.

*For the remaining steps, use a text editor to examine and modify the metadata and script files. Place copied statements into the new script file in the same order they occur in the metadata file to avoid dependency errors.*

4. Search for each instance of double quotes in the extracted metadata file. These can occur in triggers, stored procedures, views, domains, table column defaults, and constraints. Change each double quote that delimits a string to a single quote. Make a note of any tables that have column-level constraints or column defaults in double quotes.

Copy each changed statement to your empty script file, but do not copy ALTER TABLE statements whose only double quotes are in column-level constraints or column defaults.

*Important* When copying trigger or stored procedure code, be sure to include any associated SET TERM statements.

*Quoted quotes* If there is any chance that you have single or double quotes *inside* of strings, you must search and replace on a case-by-case basis to avoid inappropriate changes. The handling of quotation marks within strings is as follows:

<i>String:</i>	In "peg" mode
<i>Double-quoted:</i>	"In ""peg"" mode"
<i>Single-quoted:</i>	'In "peg" mode'
<i>String:</i>	O'Reilly
<i>Double-quoted:</i>	"O'Reilly"
<i>Single-quoted:</i>	'O' 'Reilly'

TABLE 3.6 Handling quotation marks inside of strings

5. In the new script file, search for occurrences of the `TIMESTAMP` datatype. In most cases, these were `DATE` datatypes in your pre-6 database. For each one, decide whether you want it to be `TIME`, `TIMESTAMP`, or `DATE` in your dialect 3 database. Change it as needed.
6. Repeat step 5 in the metadata file. Copy each changed statement to your new script file.
7. In the new script file, search for occurrences of reserved words that are used as object names and enclose them in double quotes; that makes them delimited identifiers.
8. Repeat step 7 in the metadata file. Copy each changed statement to your new script file.
9. In each of the two files, search for each instance of a `DECIMAL` or `NUMERIC` datatype with a precision greater than 9. Consider whether or not you want data stored in that column or with that domain to be stored as `DOUBLE PRECISION` or `INT64`. See **“Do you really need to migrate your `NUMERIC` and `DECIMAL` datatypes?” on page 48** for a discussion of issues. For occurrences that should be stored as `DOUBLE PRECISION`, change the datatype to that. Leave occurrences that you want stored as `INT64` alone for now. Copy each changed statement that occurs in the metadata file to your new script file.

*Perform the following steps in your new script file:*

10. Locate each CREATE TRIGGER and CREATE DOMAIN statement and change it to ALTER TRIGGER or ALTER DOMAIN as appropriate.
11. Locate each CREATE VIEW statement. Precede it by a corresponding DROP statement. For example, if you have a CREATE VIEW *foo* statement, put a DROP VIEW *foo* statement right before it, so that when you run this script against your database, each view first gets dropped and then re-created.
12. If you have any ALTER TABLE statements that you copied because they contain named table-level constraints, modify the statement so that it does nothing except drop the named constraint and then add the constraint back with the single quotes.
13. Check that stored procedure statements are ALTER PROCEDURE statements. This should already be the case.
14. At the beginning of the script, put a CONNECT statement that connects to the original database that you are migrating.
15. Make sure your database is backed up and run your script against the database.
16. Use **gfix** to change the database dialect to 3.

```
gfix -sql_dialect 3 database.gdb
```

**Note** To run **gfix** against a database, you must attach as either the database owner or SYSDBA.

17. At this point, DECIMAL and NUMERIC columns with a precision greater than 9 are still stored as DOUBLE PRECISION. To store the data as INT64, follow the steps in **“Migrating NUMERIC and DECIMAL datatypes” on page 48**.
18. Validate the database using either IBConsole or **gfix**.

That’s it. You’ve got a dialect 3 database. There is a little more work to do if you want your NUMERIC and DECIMAL columns with a precision of greater than 9 to be stored as INT64. At this point, they are still stored as DOUBLE PRECISION. To decide whether you want to change they way data in these columns is stored, read

In addition, there are some optional steps you can take that are described in the following sections, **“Column defaults and column constraints”** and **“Unnamed table constraints”**.

**IMPORTANT** If you ever extract metadata from the dialect 3 database that you created using the steps above, and if you plan to use that metadata to create a new database, check to see if the extracted metadata contains double quotes delimiting string constants in column defaults, column constraints, or unnamed table constraints. Change any such occurrences to single quotes before using the metadata to create the new database.

### ► *Column defaults and column constraints*

The steps above permitted you to retain double quoted string constants in column defaults, column constraints, and unnamed table constraints. This is possible because, once created, InterBase stores them in binary form.

Following the steps above creates a dialect 3 database that is fully functional, but if it contains double quoted string constants in column defaults, column constraints, or unnamed column constraints, inconsistencies are visible when you SHOW metadata or extract it. You can choose to resolve these inconsistencies by following these steps:

1. Back up the database.
2. Examine the metadata to detect each occurrence of a column default or column constraint that uses double quotes.
3. For each affected column, use the ALTER COLUMN clause of the ALTER TABLE statement to give the column a temporary name. If column position is likely to be an issue with any of your clients, change the position as well.
4. Create a new column with the desired datatype, giving it the original column name and position.
5. Use UPDATE to copy the data from old column to the new column:

```
UPDATE table_name
    SET new_col = old_col;
```

6. Drop the old column.

### ► *Unnamed table constraints*

Read the first two paragraphs under **“Column defaults and column constraints” on page 47** to understand why you don’t always need to change constraints with double quotes to single-quoted form, and why you might want to change them.

To bring unnamed table constraints that contain double quotes into compliance with the dialect 3 standard, follow these steps:

1. Back up the database.
2. Examine the metadata to detect each occurrence of an unnamed table constraint that uses double quotes.
3. For each occurrence, use SHOW TABLE to see the name that InterBase has assigned to the constraint.
4. Use ALTER TABLE to drop the old constraint, using the name given in the SHOW TABLE output and add a new constraint. For ease in future handling, give the constraint a name.

If SHOW TABLE shows that InterBase stores the unnamed constraint as “INTEG\_2”, then issue the following statement to change the constraint:

```
ALTER TABLE foo
    DROP CONSTRAINT INTEG_2,
    ADD CONSTRAINT new_name
        CHECK (col_name IN ('val1', 'val2', 'val3'));
```

### ► *Migrating NUMERIC and DECIMAL datatypes*

If you back up a NUMERIC or DECIMAL column with a precision greater than 9 (for example, NUMERIC(12,2)) in an InterBase 5 or earlier database and restore the database as InterBase 6, the column is still stored as DOUBLE PRECISION. Because InterBase does not allow datatype conversions that could potentially result in data loss, you cannot use the ALTER COLUMN statement to change the column datatype from DOUBLE PRECISION to INT64. To migrate a DOUBLE PRECISION column to an INT64 column, you must create a new INT64 column and copy the contents of the older column into it.

In InterBase 6 dialect 3, when you create a NUMERIC or DECIMAL column with a precision of greater than 9, data in it is automatically stored as an INT64 exact numeric.

If you want NUMERIC and DECIMAL datatypes with a precision greater than 9 to be stored as exact numerics, you must take some extra steps after migrating to dialect 3. The following sections tell you how to decide whether you really need to take these steps and how to perform them if you decide you want the exact numerics.

#### **DO YOU REALLY NEED TO MIGRATE YOUR NUMERIC AND DECIMAL DATATYPES?**

As you migrate your databases to dialect 3, consider the following questions about columns defined with NUMERIC and DECIMAL datatypes:

- Is the precision less than 10? If so, there is no issue. You can migrate without taking any action and there will be no change in the database and no effect on clients.
- For NUMERIC and DECIMAL columns with precision greater than 9, is DOUBLE PRECISION an appropriate way to store your data?
  - In many cases, the answer is “yes.” If you want to continue to store your data as DOUBLE PRECISION, change the datatype of the column to DOUBLE PRECISION either before or after migrating your database to dialect 3. This doesn’t change any functionality in dialect 3, but it brings the declaration into line with the storage mode. In a dialect 3 database, newly-created columns of this type are stored as INT64, but migrated columns are still stored as DOUBLE PRECISION. Changing the declaration avoids confusion.



- DOUBLE PRECISION may not be appropriate or desirable for financial applications and others that are sensitive to rounding errors. In this case, you need to take steps to migrate your column so that it is stored as INT64 in dialect 3. As you make this decision, remember that INT64 does not store the same range as DOUBLE PRECISION. Check whether you will experience data loss and whether this is acceptable.

### MIGRATING NUMERIC AND DECIMAL DATATYPES

Read **“Do you really need to migrate your NUMERIC and DECIMAL datatypes?” on page 48** to decide whether you have columns in a dialect 1 database that would be best stored as 64-bit integers in a dialect 3 database. If this is the case, follow these steps for each column:

1. Migrate your database to InterBase 6 as described in **“Method one: in-place migration” on page 44**.
2. Use the ALTER COLUMN clause of the ALTER DATABASE statement to change the name of each affected column to something different from its original name. If column position is going to be an issue with any of your clients, use ALTER COLUMN to change the positions as well.
3. Create a new column for each one that you are migrating. Use the original column names and if necessary, positions. Declare each one as a DECIMAL or NUMERIC with precision greater than 9.
4. Use UPDATE to copy the data from each old column to its corresponding new column:

```
UPDATE tablename
    SET new_col = old_col;
```

5. Check that your data has been successfully copied to the new columns and drop the old columns.

---

### Method two: migrating to a new database

If you can create a data transfer utility that copies data between databases, the process of migrating a database to dialect 3 is considerably simplified.

**Overview** Extract the metadata from your database, examine it for problem areas, and fix the problems. Use the modified metadata file to create a new dialect 3 database and use an application to transfer data from the old database to the new.

1. If you have not migrated the database to version 6, dialect 1, do so first. Back up the database again.
2. Extract the metadata from the database using `isql -x`. If you are migrating a database that contains data structures created with GDML, see **“Migrating older databases” on page 50**.

*For the following steps, use a text editor to examine and modify the metadata file.*

3. Search for each occurrence of the `TIMESTAMP` datatype. In most cases, these were `DATE` datatypes in your pre-6 database. Decide whether you want it to be `TIME`, `TIMESTAMP`, or `DATE` in your dialect 3 database. Change it as needed.
4. Find all instances of reserved words that are used as object names and enclose them in double quotes to make them delimited identifiers.
5. Search for each instance of double quotes in the extracted metadata file. These can occur in triggers, stored procedures, views, domains, exceptions, table column defaults, and constraints. Change each double quote to a single quote.
6. Search for each instance of a `DECIMAL` or `NUMERIC` datatype with a precision greater than 9. Consider whether or not you want that data stored as `DOUBLE PRECISION` or `INT64`. See **“Do you really need to migrate your `NUMERIC` and `DECIMAL` datatypes?” on page 48** for a discussion of issues. For occurrences that should be stored as `DOUBLE PRECISION`, change the datatype to that. Leave occurrences that you want stored as `INT64` alone for now.
7. At the beginning of the file, enter `SET SQL DIALECT 3`. On the next line, uncomment the `CREATE DATABASE` statement and edit it as necessary to create a new database.
8. Run the metadata file as a script to create a new database.
9. Use your data transfer utility to copy data from the old database to the new dialect 3 database. In the case of a large database, allow significant time for this.
10. Validate the database using `gfix`.
11. At this point, `DECIMAL` and `NUMERIC` columns with a precision greater than 9 are still stored as `DOUBLE PRECISION`. To store the data as `INT64`, follow the steps in **“Migrating `NUMERIC` and `DECIMAL` datatypes” on page 48**.

### ► *Migrating older databases*

If you have legacy databases in which some data structures were created with GDML, you may need to extract metadata in a slightly different way.

1. Try extracting metadata as described in Step 2 above and examine it to see if all tables and other DDL structures are present. If they are not, delete the metadata file and extract using the **-a** switch instead of the **-x** switch. This extracts objects created in GDML.
2. You may have to change some of the code to SQL form. For example, the following domain definition

```
CREATE DOMAIN NO_INIT_FLAG AS SMALLINT
  ( no_init_flag = 1 or
    no_init_flag = 0 or
    no_init_flag missing);
```

needs to be translated to:

```
CREATE DOMAIN NO_INIT_FLAG AS SMALLINT
  CHECK ( VALUE = 1 OR VALUE = 0 OR VALUE IS NULL );
```

3. Some code may be commented out. For example:

```
CREATE TABLE BOILER_PLATE (BOILER_PLATE_NAME NAME,
  DATE DATE,
  CREATED_DATE COMPUTED BY /* Date */);
```

needs to be changed to:

```
CREATE TABLE BOILER_PLATE (BOILER_PLATE_NAME NAME,
  "DATE" DATE,
  CREATED_DATE COMPUTED BY "DATE");
```

---

## Migrating clients

To migrate an older client application to InterBase 6, install the InterBase 6 client onto the platform where the client application resides. An InterBase server then recognizes that client as a version 6 dialect 1 client.

It is good practice to recompile and relink the application and make note of field names, datatype use, and so on in the new application. When you recompile, state the dialect explicitly:

```
SET SQL DIALECT n;
```

**IMPORTANT** If you have databases that use any of the new version 6 keywords as object identifiers and you are not migrating those databases to dialect 3, you might want to not migrate your version 5 clients. If you migrate them to version 6 dialect 1, you lose the ability to access those keyword columns. See **“New keywords” on page 27**.

When you recompile an existing **gpre** client, you must recompile it with the **gpre -sql\_dialect *n*** switch.

There are several paths that permit you to create dialect 3 clients that access all new InterBase 6 features:

- In Delphi 5, make calls to functions in the new InterBase Express (IBX) package. Because the Delphi 5 beta includes InterBase 5, it ships with a version of IBX that does not include calls to the new InterBase 6 Services, Install, and Licensing APIs.
- To write embedded SQL applications that address all InterBase 6 dialect 3 functionality, compile them using **gpre -sql\_dialect 3**.

Client	How to migrate
Older applications such as InterBase version 5 applications	<ul style="list-style-type: none"> <li>• Dialect is 1; there is no way to change the dialect</li> <li>• A version 5 client application becomes version 6 dialect 1 client whenever the InterBase 6 client is installed on the machine with the client</li> </ul>
ISQL	<ul style="list-style-type: none"> <li>• Issue the command line option: -sql_dialect <i>n</i></li> <li>• Or issue the command SET SQL DIALECT <i>n</i>;</li> </ul>
GPRE	<ul style="list-style-type: none"> <li>• Issue the command line option -sql_dialect <i>n</i></li> <li>• Or issue the command EXEC SQL SET SQL DIALECT <i>n</i>;</li> </ul>
BDE	All applications use SQL dialect 1. To access InterBase dialect 3 features from Delphi 5, use the IBX components
InterClient	All applications use SQL dialect 1
Direct API calls	Set the dialect parameter on <i>isc_dsql_execute_immediate()</i> , <i>isc_dsql_exec_immed2()</i> , <i>isc_dsql_prepare()</i> API calls to the desired dialect value: 1 or 3

TABLE 3.7 Migrating clients: summary

---

## IBReplicator migration issues

InterBase 6 contains a new version of IBReplicator that should be used instead of previous versions. It contains new features, described in the *Release Notes* and in the *Operations Guide*, and a few changes which should be addressed when moving from InterBase 5 to InterBase 6. If you have been using IBReplicator with previous versions of InterBase, keep these issues in mind:

- If you have any schemas defined that have a source database replicating to more than one target (within the same schema), then you should run the **Create System Objects** command for each of those source databases. In such schemas, more than one entry is placed in the log for any row modified. This does not cause any data errors, but does cause some changes to be replicated more than once.

**Note** Do not run the **Remove System Objects** command, as this will empty the REPL\_LOG table.

- If you have been using older licenses purchased from Synectics Software, those licenses will not work with InterBase 6. You must use the version of IBReplicator for Opensource InterBase, or buy new licenses from Borland Software Corporation for use with the version of IBReplicator for Borland InterBase (the certified commercial version of InterBase).

---

## Migrating data from other DBMS products

If you have a large amount of data in another DBMS such as Paradox, the most efficient way to bring the data into InterBase is to export the data from the original DBMS into InterBase external file format. (See the *Data Definition Guide* for more information about InterBase external files.) Then insert the data from the external files into the internal tables. It is best not to have any constraints on new internal tables; you can validate the database more easily once the data is in InterBase. If constraints do exist, you will need triggers to massage the incoming data.

