# New in Firebird 2

# New features implemented

This chapter summarizes the new features implemented in Firebird 2, encompassing both v.2.1 and the v.2.0.x series.

Important change to API DPB parameters in v.2.1.2: A long-standing, legacy loophole in the handling of DPB parameters enabled ordinary users to make connection settings that could lead to database corruptions or give them access to SYSDBA-only operations. The loophole has been closed, a change that could affect several existing applications, database tools and connectivity layers (drivers, components). Details are in Chapter 3, Changes to the Firebird API and ODS.

### On-disk structure

Databases created or restored under Firebird 2 have an on-disk structure (ODS) of 11 or higher.

1. Firebird 2.1 creates databases with an ODS of 11.1. It can read databases of lower ODS but most of its new features will be unavailable to such databases.
2. Firebird 2.0.x servers create databases with an ODS of 11 (sometimes expressed as 11.0). If you wish to have the full range of v.2.1 features available, you should upgrade ODS 11 and lower databases by backing them up and restoring them under v.2.1.

### Database triggers

(**v.2.1**) Newly implemented database triggers are user-defined PSQL modules that can be designed to fire in various connection-level and transaction-level events. See Database triggers.

### SQL and objects

### Global temporary tables

(**v.2.1**) SQL standards-compliant global temporary tables have been implemented. These pre-defined tables are instantiated on request for connection-specific or transaction-specific use with non-persistent data, which the Firebird engine stores in temporary files. See Global temporary tables.

### Common table expressions, recursive DSQL queries

(**v.2.1**) Standards-compliant common table expressions, which make dynamic recursive queries possible, are introduced. See Common table expressions.

### RETURNING clause

(**v.2.1**) Optional RETURNING clause for all singleton operations update, insert and delete operations. See The RETURNING clause.

### UPDATE OR INSERT statements

(**v.2.1**) Now you can write a statement that is capable of performing either an update to an existing record or an insert, depending on whether the targeted record exists. See UPDATE OR INSERT statement.

### MERGE statement

(**v.2.1**) New statement syntax that performs either an update to an existing record if a condition is met or an insert if the condition is not met. See MERGE statement.

### LIST() function

(**v.2.1**) A new aggregate function LIST(<SOMETHING>) retrieves all of the SOMETHINGs in a group and aggregates them into a comma-separated list. See LIST function.

### Lots of new built-in functions

(**v.2.1**) Built-in functions replacing many of the UDFs from the Firebird-distributed UDF libraries. For a full list with examples, see Built-in functions.

### "Short" BLOBs can masquerade as long VARCHARs

(**v.2.1**) At various levels of evaluation, the engine now treats text BLOBs that are within the 32,765-byte size limit as though they were varchars. Now functions like cast, lower, upper, trim and substring will work with these BLOBs, as well as concatenation and assignment to string types. See Text BLOB compatibility.

*Important note about use of* SUBSTRING() *with a* BLOB: The SUBSTRING() function now returns a BLOB, not a VARCHAR as previously.

back to top of page

### Procedural SQL

### Domains for defining PSQL variables and arguments

(**v.2.1**) PSQL local variables and input and output arguments for stored procedures can now be declared using domains in lieu of canonical data types. See Domains in PSQL.

### NOT NULL constraint on stored procedure parameters and variables

(**v.2.1**) Input and output arguments and PSQL local variables can be constrained as non-nullable, using the optional keywords NOT NULL in their declarations. See NOT NULL in stored procedures.

### COLLATE in stored procedures and parameters

(**v.2.1**) Collations can now be applied to PSQL variables and arguments. See COLLATE in stored procedures.

### Enhancement to PSQL error stack trace

V. Khorsun

Feature request CORE-970

(**v.2.1**) A PSQL error stack trace now shows line and column numbers.

back to top of page

## Security

### Windows security to authenticate users

(**v.2.1**) Windows “Trusted User” security can be applied for authenticating Firebird users on a Windows host. See Windows trusted user security.

back to top of page

## International language support

### The CREATE COLLATION command

(**v.2.1**) The DDL command CREATE COLLATION has been introduced for implementing a collation, obviating the need to use the script for it. See CREATE COLLATION statement.

### Unicode collations anywhere

(**v.2.1**) Two new Unicode collations can be applied to any character set using a new mechanism. See UNICODE collations.

back to top of page

## Platform support

### Ports to Windows 2003 64-bit

D. Yemanov

**Feature request** CORE-819 **and** CORE-682

(**v.2.1**) 64-bit Windows platform (AMD64 and Intel EM64T) ports of Classic, Superserver and Embedded models.

back to top of page

## Administrative

### Database monitoring via SQL

(**v.2.1**) Implementation of run-time database snapshot monitoring (transactions, tables, etc.) via SQL over some new virtualized system tables. See Monitoring tables.

Included in the set of tables is one named MON$DATABASE that provides a lot of the database header information that could not be obtained previously via SQL: such details as the on-disk structure (ODS) version, SQL dialect, sweep interval, OIT and OAT and so on.

It is possible to use the information from the monitoring tables to cancel a rogue query. See Cancel a running query.

### More context information

Context information providing the server engine version has been added, for retrieving via SELECT calls to the RDB$GET_CONTEXT function. See More context information.

### New command-line utility fbsvcmgr

(**v.2.1**) The new utility fbsvcmgr provides a command-line interface to the Services API, enabling access to any service that is implemented in Firebird.

Although there are numerous database administration tools around that surface the Services API through graphical interfaces, the new tool addresses the problem for admins needing to access remote Unix servers in broad networks through a text-only connection. Previously, meeting such a requirement needed a programmer. Details here.

back to top of page

#### Remote interface

(**v.2.1**) The remote protocol has been slightly improved to perform better in slow networks once drivers are updated to utilise the changes. Testing showed that API round trips were reduced by about 50 percent, resulting in about 40 per cent fewer TCP round trips. See Remote interface improvement.

### Derived tables

A. Brinkman

Implemented support for derived tables in DSQL (subqueries in FROM clause) as defined by SQL200X. A derived table is a set, derived from a dynamic SELECT statement. Derived tables can be nested, if required, to build complex queries and they can be involved in joins as though they were normal tables or views.

More details under Derived tables in the DML chapter.

### PSQL now supports named cursors

D. Yemanov

Multiple named (i.e. explicit) cursors are now supported in PSQL and in DSQL EXECUTE BLOCK statements. More information in the PSQL chapter, Explicit cursors.

back to top of page]]

### Reimplemented protocols on Windows

D. Yemanov

Two significant changes have been made to the Windows-only protocols.

### Local Protocol--XNET

Firebird 2.0 has replaced the former implementation of the local transport protocol (often referred to as IPC or IPServer) with a new one, named XNET.

It serves exactly the same goal, to provide an efficient way to connect to a server located on the

same machine as the connecting client without a remote node name in the connection string. The new implementation is different and addresses the known issues with the old protocol.

Like the old IPServer implementation, the XNET implementation uses shared memory for inter-process communication. However, XNET eliminates the use of window messages to deliver attachment requests and it also implements a different synchronization logic.

#### Benefits of the XNET Protocol over IPServer

Besides providing a more robust protocol for local clients, the XNET protocol brings some notable benefits:

- it works with Classic Server
- it works for non-interactive services and terminal sessions
- it eliminates lockups when a number of simultaneous connections are attempted.

#### Performance

The XNET implementation should be similar to the old IPServer implementation, although XNET is expected to be slightly faster.

#### Disadvantages

The one disadvantage is that the XNET and IPServer implementations are not compatible with each other. This makes it essential that your fbclient.dll version should match the version of the server binaries you are using (fbserver.exe or fb_inet_server.exe) exactly. It will not be possible to to establish a local connection if this detail is overlooked. (A TCP localhost loopback connection via an ill-matched client will still do the trick, of course).

### Change to WNET ("NetBEUI") Protocol

WNET (a.k.a. NetBEUI) protocol no longer performs client impersonation.

In all previous Firebird versions, remote requests via WNET are performed in the context of the *client security token*. Since the server serves every connection according to its client security credentials, this means that, if the client machine is running some OS user from an NT domain, that user should have appropriate permissions to access the physical database file, UDF libraries, etc., on the server file system. This situation is contrary to what is generally regarded as proper for a client-server setup with a protected database.

Such impersonation has been removed in Firebird 2.0. WNET connections are now truly client-server and behave the same way as TCP ones, i.e., with no presumptions with regard to the rights of OS users.

back to top of page

### Reworking of garbage collection

V. Khorsun

Since Firebird 1.0 and earlier, the Superserver engine has performed background garbage collection, maintaining information about each new record version produced by an UPDATE or DELETE statement. As soon as the old versions are no longer “interesting”, i.e. when they become older than the Oldest Snapshot transaction (seen in the gstat -header output) the engine signals for them to be removed by the garbage collector.

Background GC eliminates the need to re-read the pages containing these versions via a SELECT COUNT(*) FROM aTable or other table-scanning query from a user, as occurs in Classic and in versions of InterBase prior to v.6.0. This earlier GC mechanism is known as cooperative garbage collection.

Background GC also averts the possibility that those pages will be missed because they are seldom read. (A sweep, of course, would find those unused record versions and clear them, but the next sweep is not necessarily going to happen soon.) A further benefit is the reduction in I/O, because of the higher probability that subsequently requested pages still reside in the buffer cache.

Between the point where the engine notifies the garbage collector about a page containing unused versions and the point when the garbage collector gets around to reading that page, a new transaction could update a record on it. The garbage collector cannot clean up this record if this later transaction number is higher than the Oldest Snapshot or is still active. The engine again notifies the garbage collector about this page number, overriding the earlier notification about it and the garbage will be cleaned at some later time.

In Firebird 2.0 Superserver, both cooperative and background garbage collection are now possible. To manage it, the new configuration parameter GCPolicy was introduced. It can be set to:

- cooperative - garbage collection will be performed only in cooperative mode (like Classic) and the engine will not track old record versions. This reverts GC behaviour to that of IB 5.6 and earlier. It is the only option for Classic.
- background - garbage collection will be performed only by background threads, as is the case for Firebird 1.5 and earlier. User table-scan requests will not remove unused record versions but will cause the GC thread to be notified about any page where an unused record version is detected. The engine will also remember those page numbers where UPDATE and DELETE statements created back versions.
- combined (the installation default for Superserver) - both background and cooperative garbage collection are performed.

*Note:* The Classic server ignores this parameter and always works in cooperative mode.

back to top of page

### Porting of the Services API to Classic is complete

N. Samofatov

Porting of the Services API to Classic architecture is now complete. All Services API functions are now available on both Linux and Windows Classic servers, with no limitations. Known issues with gsec error reporting in previous versions of Firebird are eliminated.

### Lock timeout for WAIT transactions

A. Karyakin, D. Yemanov

All Firebird versions provide two transaction wait modes: NO WAIT and WAIT. NO WAIT mode means that lock conflicts and deadlocks are reported immediately, while WAIT performs a blocking wait which times out only when the conflicting concurrent transaction ends by being committed or rolled back.

The new feature extends the WAIT mode by making provision to set a finite time interval to wait for the concurrent transactions. If the timeout has passed, an error (isc_lock_timeout) is reported.

Timeout intervals are specified per transaction, using the new TPB constant isc_tpb_lock_timeout in the API or, in DSQL, the LOCK TIMEOUT <value> clause of the SET TRANSACTION statement.

back to top of page

### New implementation of string search operators

N. Samofatov

1. The operators now work correctly with BLOBs of any size. Issues with only the first segment being searched and with searches missing matches that straddle segment boundaries are now gone.
2. Pattern matching now uses a single-pass Knuth-Morris-Pratt algorithm, improving performance when complex patterns are used.
3. The engine no longer crashes when NULL is used as ESCAPE character for LIKE.

back to top of page

### Reworking of updatable views

D. Yemanov

A reworking has been done to resolve problems with views that are implicitly updatable, but still have update triggers. This is an important change that will affect systems written to take advantage of the undocumented [mis]behaviour in previous versions.

For details, see the notes in the *Compatibility* chapter of the separate *Installation Notes* document.

### Additional database shutdown modes introduced

N. Samofatov

Single-user and full shutdown modes are implemented using new [state] parameters for the gfix -shut and gfix -online commands.

### Syntax Pattern

```
gfix <command> [<state>] [<options>]

<command>> ::= {-shut | -online}
<state> ::= {normal | multi | single | full}
<options> ::= {-force <timeout> | -tran | -attach}
```

- **normal state:** online database.
- **multi state:** multi-user shutdown mode (the legacy one, unlimited attachments of SYSDBA/owner are allowed).
- **single state:** single-user shutdown (only one attachment is allowed, used by the restore process).
- **full state:** full/exclusive shutdown (no attachments are allowed).

For more details, refer to the section on Gfix new shutdown modes, in the Utilities chapter.

For a list of shutdown state flag symbols and an example of usage, see Shutdown state in the API.

back to top of page

### UDFs improved re NULL handling

C. Valderrama

### Signalling SQL NULL

- Ability to signal SQL NULL via a NULL pointer (see Signal SQL NULL in UDFs).
- External function library ib_udf upgraded to allow the string functions ASCII_CHAR, LOWER, LPAD, LTRIM, RPAD, RTRIM, SUBSTR and SUBSTRLEN to return NULL and have it interpreted correctly.

The script ib_udf_upgrade.sql can be applied to pre-v.2 databases that have these functions declared, to upgrade them to work with the upgraded library. This script should be used only when you are using the new ib_udf library with Firebird v2 and operation requests are modified to anticipate nulls.

### Run-time checking for concatenation overflow

D. Yemanov

Compile-time checking for concatenation overflow has been replaced by run-time checking.

From Firebird 1.0 onward, concatenation operations have been checked for the possibility that the resulting string might exceed the string length limit of 32,000 bytes, i.e. overflow. This check was performed during the statement prepare, using the declared operand sizes and would throw an error for an expressions such as:

```
CAST('qwe' AS VARCHAR(30000)) || CAST('rty' AS VARCHAR(30000))
```

From Firebird 2.0 onward, this expression throws only a warning at prepare time and the overflow check is repeated at runtime, using the sizes of the actual operands. The result is that our example will be executed without errors being thrown. The isc_concat_overflow exception is now thrown only for actual overflows, thus bringing the behaviour of overflow detection for concatenation into line with that for arithmetic operations.

back to top of page

### Changes to synchronisation logic

N. Samofatov

- Lock contention in the lock manager and in the Superserver thread pool manager has been reduced significantly.
- A rare race condition was detected and fixed, that could cause the Superserver to hang during request processing until the arrival of the next request.
- Lock manager memory dumps have been made more informative and OWN_hung is detected correctly.
- Decoupling of lock manager synchronization objects for different engine instances was implemented.

### Support for 64-bit platforms

A. Peshkov, N. Samofatov

Firebird 2.0 will support 64-bit platforms.

back to top of page

### Record enumeration limits increased

N. Samofatov

40-bit (64-bit internally) record enumerators have been introduced to overcome the ~30GB table size limit imposed by 32-bit record enumeration.

### Debugging improvements

Various contributors

### Improved reporting from bugchecks

BUGCHECK log messages now include file name and line number. (A. Brinkman)

### Updated iInternal structure reporting

Routines that print out various internal structures (DSQL node tree, BLR, DYN, etc) have been updated. (N. Samofatov)

### New debug logging facilities

Thread-safe and signal-safe debug logging facilities have been implemented. (N. Samofatov)

### Diagnostic enhancement

Syslog messages will be copied to the user's tty if a process is attached to it. (A. Peshkov)

back to top of page

### Improved connection handling on POSIX Superserver

A. Peshkov

Posix SS builds now handle SIGTERM and SIGINT to shutdown all connections gracefully. (A. Peshkov)

### PSQL invariant tracking reworked

N. Samofatov

Invariant tracking in PSQL and request cloning logic were reworked to fix a number of issues with recursive procedures, for example SF bug #627057.

Invariant tracking is the process performed by the BLR compiler and the optimizer to decide whether an “invariant” (an expression, which might be a nested subquery) is independent from the parent context. It is used to perform one-time evaluations of such expressions and then cache the result.

If some invariant is not determined, we lose in performance. If some variant is wrongly treated as invariant, we see wrong results.

### Example

```
select * from rdb$relations
  where rdb$relation_id <
    ( select rdb$relation_id from rdb$database )
```

This query performs only one fetch from rdb$database instead of evaluating the subquery for every row of rdb$relations.

back to top of page

### ROLLBACK RETAIN syntax support

D. Yemanov

Firebird 2.0 adds an optional RETAIN clause to the DSQL ROLLBACK statement to make it consistent with COMMIT [RETAIN].

See ROLLBACK RETAIN syntax in the chapter about DML.

### No more Registry search on Win32 servers

D. Yemanov

The root directory lookup path has changed so that server processes on Windows no longer use the Registry.

*Important:* The command-line utilities still check the Registry.

### More optimizer improvements

A. Brinkman

Better cost-based calculation has been included in the optimizer routines.