Firebird Configuration Reference

Mark Rotteveel

Version 0.7, 23 December 2025

About this manual

Firebird is a (SQL) relational database management system. In simple use-cases, you can install it and use its default configuration. For more advanced use-cases, or when you want to tune Firebird to get the best performance, you will need to dig into its configuration options.

This manual is a reference for all configuration options that Firebird exposes in its configuration files, and as environment variables.

If you're missing anything or want to make suggestions for improvements, do not hesitate to contact us on firebird-devel or through the firebird-documentation issue tracker.

Firebird versions

This document covers Firebird 4.0 and Firebird 5.0. Our intent is to keep it up-to-date with new, deprecated or removed configuration options as versions are released.

We do not document configuration options for Firebird 3.0 or older if they no longer exist in Firebird 4.0.

Conventions

This manual covers multiple versions. New Firebird versions may add new configuration options, deprecate configuration options (i.e. use is discouraged but still supported), and even remove configuration options entirely.

For each configuration option, we document the version that added, deprecated or removed an option.

For example, say an option is added in Firebird 4.0, deprecated in 5.0, and removed in 6.0, that would be shown as:

Added 4.0

Deprecated 5.0

Removed 6.0

If an option is added/deprecated/removed in a point release, the point-release version (e.g. 4.0.3) will be used.

For an option which is added in Firebird 5.0, and not deprecated and removed, it just shows:

Added 5.0

If an option already existed in Firebird 3.0 or older, we do not show an *Added* item. However, if an option was deprecated in an older version, and still exists in Firebird 4.0, it is listed with the version that deprecated it.

For example:

Deprecated 2.0

Options which were removed with or before Firebird 4.0.0 are not documented.

Where to get more information

For news and information on Firebird, visit https://firebirdsql.org/.

For more Firebird documentation, go to:

- Firebird RDBMS Documentation
  - Release Notes
  - Reference Manuals
    - Firebird 5.0 Language Reference
    - Firebird 4.0 Language Reference
    - …see full list on Reference Manuals…
  - …see more options on Firebird RDBMS Documentation…

If you're looking for help on Firebird, subscribe to firebird-support.
Other mailing lists and groups can be found on Mailing Lists.

Reporting errors or missing content

If you find errors, missing content or other problems in this document, please report this in our issue tracker of the firebird-documentation GitHub repository.

Pull requests with changes and fixes are also much appreciated.

Contributing

There are several ways you can contribute to the documentation of Firebird, or Firebird in general:

- Participate on the mailing lists
- Report bugs or submit pull requests on the FirebirdSQL GitHub Project
- Become a developer (contact us on firebird-devel)
- Donate to the Firebird Foundation
- Become a paying member or sponsor of the Firebird Foundation

Reference

Chapter 1. An overview of Firebird configuration

The configuration of Firebird is distributed over a number of files and other configuration methods. In broad strokes, the configuration is split over:

`firebird.conf`
: Global configuration of Firebird server and/or the client library

`databases.conf`
: Database aliases, and per-database configuration of Firebird server and/or client library

`plugins.conf`
: Plugin configuration

`replication.conf`
: Global and per-database replication configuration

Trace configuration
: Global and per-database trace configuration

Environment variables
: Environment variables to override parts of server or client configuration

The following chapters will cover each file in detail.

1.1. Shared syntax of configuration files

The configuration files (`*.conf`) share a common syntax, though not all files use all available syntax options.

1.1.1. Key-value

In its most basic form, configuration is expressed as a list of key-value pairs, or — in some files — keyword-value pairs:

Syntax

```
key-name = value
```

Example

```
DatabaseAccess = None
```

Allowed *values* depend on the specific key, but broadly has the following types:

integer

Integer values, or integers followed by `K` (kilo: 1024), `M` (mega: 1024^2 or 1,048,576) or `G` (giga: 1024^3 or 1,073,741,824).

Boolean

Truth values, where `0` is false and `1` (and any other non-zero integer value) is true. Strings `true`, `y` and `yes` are also interpreted as true, everything else as `false`.

We recommend only using `0`/`1` *or* `true`/`false` for Boolean values.

Currently, a non-numeric suffix of an integer prefix is ignored, so `123text` is interpreted as 123 → true, not false. This is an implementation artefact that may change in the future.

string

String values; individual properties may require a specific syntax in their string values, or a limited set of values, etc.

list

List of values; a list is a string value with values separated by a space (‘ ’), comma (‘,’), or semicolon (‘;’). The order is usually significant. Some lists may use only a subset of separators; this will be mentioned explicitly if relevant.

scoped values

Some configuration files (`databases.conf`, `plugins.conf`, `replication.conf`, trace configuration) allow or require keys with a string value followed by additional key-value pairs enclosed in braces.

See Scope for more information.

A (non-scoped) value or the “simple value” of a scoped value can optionally be enclosed in double quotes. The entire value must be quoted, and anything other than a comment or whitespace after the closing double quote is considered an error. This can — for example — be used if the value contains a ‘#’ (e.g. `Key = "some#value"`), or braces (‘{’/‘}’). Unquoted, the ‘#’ is interpreted as the start of a comment, and the braces as the start or end of a scoped value.

For braces, it’s also possible to escape them by doubling them ({{/}}); doubled braces are **always** undoubled, even if the value is enclosed in double quotes.

1.1.2. Scope

Some configuration files use scopes. Scopes are key-value pairs or keyword-value pairs enclosed in braces (‘{’ and ‘}’). Their exact usage and syntax depend on the specific configuration file.

In general, the syntax is:

Key or keyword + scoped value syntax

```
<key-or-keyword> [ = <simple-value> ]
{
  { <key> = <simple-value> <new-line> }...
```

```
}
```

We call the part after the = (equals) a *scoped value*[1].

Some configuration files allow a scoped value without a simple value; this — for example — establishes a default configuration. In that case, both the equals ('=') and the simple value are absent.

For example, `databases.conf` has key-value pairs, where the value is either a string (simple value), or a scoped value (string value + scope):

Scoped value in `databases.conf`

```
# key employee with simple value
employee = $(dir_sampleDb)/employee.fdb

# key security.db with scoped value
security.db = $(dir_secDb)/security4.fdb
{
    RemoteAccess = false
    DefaultDbCachePages = 256
}
```

In this example, `security.db` is the *key*, and the string `$(dir_secDb)/security4.fdb` **and** the key-value pairs between braces are the value of `security.db`. The configuration options `RemoteAccess` and `DefaultDbCachePages` are applied specifically to the security database (alias `security.db`).

As another example, *trace configuration*, `plugins.conf` and `replication.conf` have keyword-value pairs, where the value is (always) a scoped value

Scoped value in `replication.conf`*:*

```
database = /your/db.fdb
{
  journal_source_directory = /your/db/source
}
```

Here `database = /your/db.fdb` identifies this as replication configuration specific to the database `/your/db.fdb`, with the actual configuration listed within the braces.

1.1.3. Comments

A '#' starts a comment, until the end of that line. Comments are ignored by the configuration parser.

Comments are used in the default configuration files for documentation, and to show default values without explicitly configuring those values.

For your own use, you can use comments to temporarily disable a configuration item or revert to the default. You can also use comments to record why a specific configuration value was chosen

(e.g. by specifying a rationale, or a link to an external document, etc.).

If you need to specify a value that contains the '#' symbol, enclose the value in double quotes (e.g. `Key = "some#value"`).

1.1.4. Macro substitution

Firebird provides a number of predefined macros which can be used in configuration values to substitute directory locations.

The syntax to use these macros is `$(macro-name)` (e.g. `$(dir_plugins)`).

It is not possible to define your own custom macros.

Available configuration macros

root	root of the Firebird instance
install	directory where Firebird is installed
this	directory containing the current configuration file
dir_conf	directory containing `firebird.conf` and `databases.conf`
dir_secDb	directory containing the default security database
dir_plugins	directory containing the plugins
dir_udf	default directory containing UDFs
dir_sample	example directory
dir_sampleDb	directory containing the example DB (`employee.fdb`)
dir_intl	directory containing international modules
dir_msg	directory containing the messages file (`firebird.msg`)

The macro names are case-insensitive.

For replication, additional macros are available for specific configuration items; these are covered in `replication.conf`.

1.1.5. Include

The `include` statement allows you to include the content of another file when the configuration file is read.

Syntax

```
include path-expression
```

Where *path-expression* is a string identifying the file or files to include. The *path-expression* supports:

- Relative paths (resolved against the parent directory of the current configuration file)
- Absolute paths
- Macro substitution
- Wildcards (* for zero or more characters, and ? for one character)

Examples

```
# Relative path
include some_file.conf

# Which is equivalent to
include $(this)/some_file.conf

# Absolute path (Windows)
include C:\Firebird\default.conf

# Absolute path (Linux)
include /opt/firebird/default.conf

# Path with macro substitution
include $(dir_plugins)/some_file.conf

# Wildcard to include multiple files
include databases/*.conf
```

On Windows you can use either \ or / as the path separator. On Linux you can only use /.

[1] The term *scoped value* is specific to this documentation, other Firebird documentation do not use this name (yet)

Chapter 2. firebird.conf

The firebird.conf configuration file serves three purposes:

1. Configuring the Firebird server process
2. Configuring client connections made by the Firebird server

   For example, for connections made with EXECUTE STATEMENT … ON EXTERNAL

3. Configuring client connections made by other processes.

   This is the general case of the second item.

The third item does have a big caveat. The firebird.conf in the Firebird installation directory will not configure **all** client connections, only those client connections which

1. use fbclient.dll/libfbclient.so, and
2. where the *fbclient* library actually reads that firebird.conf file.

If your application uses its own *fbclient*, it will read and use the client configuration in the firebird.conf in the same directory as that DLL (Windows), or the directory above it (Linux and other OSes), *if it exists*; it will not use the firebird.conf in a Firebird installation directory[1].

The configuration used by a client can also be overridden using the isc_dpb_config DPB item or isc_spb_config SPB item. Its contents are key-value pairs as used in firebird.conf, and their values take precedence over the client configuration in firebird.conf.

Some server-side configuration items in firebird.conf can also be configured per database in databases.conf. The value in databases.conf takes precedence over the value in firebird.conf, which takes precedence over the default value. If a value is user-configurable (e.g. through the database parameter buffer (DPB) or a SET statement), that value will generally take precedence for the current connection, though in some cases the server configuration may establish an upper limit (e.g. for timeouts).

Since Firebird 4.0, it is possible to query configuration settings of the current database for the current connection from the RDB$CONFIG virtual table:

```
select * from RDB$CONFIG
```

2.1. Configuration items

The following sections list the configuration items supported in firebird.conf. If a configuration item is also valid in databases.conf, this is mentioned explicitly. Unless mentioned otherwise, configuration items are server-side only.

Where applicable, we'll report the version that a configuration item was added, deprecated, or removed; see also Conventions.

The configuration items are listed in order of appearance in the default `firebird.conf` file.

2.1.1. DatabaseAccess

Configures the database paths — other than aliases — the server accepts for opening databases.

Configuration

Global

Syntax

```
DatabaseAccess = <database-access-config>

<database-access-config> ::=
    None
  | Full
  | Restrict <path-list>

<path-list> ::= path [; path ...]
```

Default

`Full` (see Security recommendation!)

Supported values

`None`

Firebird only accepts database aliases defined in `databases.conf`.

`Full`

Firebird accepts database aliases, and all paths accessible to the Firebird server (i.e. read/write access, or read access for a read-only database).

`Restrict <path-list>`

Firebird accepts database aliases, and databases with a path rooted in the directories listed in `<path-list>` (that is, the database file is in a listed directory or in a subdirectory of a listed directory).

The `<path-list>` is a semicolon-separated list of directories. It is possible to use absolute paths (e.g. Windows — `C:\Database`, Linux — `/db`), and relative paths. Relative paths are resolved against the root directory of Firebird. Given relative paths are not always obvious, it is recommended to use absolute paths.

If environment variable `ISC_PATH` is not set, the paths listed are used to resolve filenames **without** a path component, including a check for existence. The first entry in the list is used as the fallback if no such file exists.

Security recommendation

The default value of `Full` is not recommended for production systems. Full database access may compromise your system[2].

> The most secure option is to use None, so all databases must be listed in databases.conf. If your users or applications must be able to dynamically create databases, then use Restrict with a list of allowed database paths.
>
> We also recommend that there is no overlap with the directories listed in ExternalFileAccess. That is, directories listed in DatabaseAccess should not be the same as, be contained in, or contain the directories listed in ExternalFileAccess. See ExternalFileAccess for more information.

Examples

```
# Only allow aliases
DatabaseAccess = None
# Full access
DatabaseAccess = Full
# Restricted (Windows)
DatabaseAccess = Restrict C:\Databases;D:\CRM
# Restricted (Linux)
DatabaseAccess = Restrict /var/db;/var/crm
```

See also

ExternalFileAccess, ISC_PATH

2.1.2. RemoteAccess

Enables remote access of databases.

Configuration

Global, and per-database

Syntax

```
RemoteAccess = Boolean
```

Default

true

The Boolean option RemoteAccess controls if databases can be opened remotely — via TCP/IP, or WNET[3], or only locally using *embedded* or — on Windows — XNET.

Any TCP/IP or WNET connection is considered remote, including connections created from the same machine (localhost).

> **Security recommendation**
>
> The security database (securityN.fdb, alias security.db) has RemoteAccess set to false in databases.conf in a standard Firebird installation. If you use additional dedicated security databases, we recommend disabling remote access for them too.

> If you have high security requirements, consider setting RemoteAccess to false in firebird.conf to disallow remote access for all databases, and enabling it per database in databases.conf as needed.

Examples

```
# Enable remote access
RemoteAccess = true
# Disable remote access
RemoteAccess = false
```

2.1.3. ExternalFileAccess

Configures paths allowed for external table files.

Configuration

Global, and per-database

Syntax

```
ExternalFileAccess = <external-file-access-config>

<external-file-access-config> ::=
    None
  | Full
  | Restrict <path-list>

<path-list> ::= path [; path ...]
```

Default

None

Supported values

None

No paths are allowed as external table file.

Full

All paths are allowed as an external table file.

Restrict <path-list>

Files with a path rooted in the directories listed in <path-list> (that is, the file is in a listed directory or in a subdirectory of a listed directory) are allowed as an external table file.

The <path-list> is a semicolon-separated list of directories. It is possible to use absolute paths (e.g. Windows — C:\ExternalFiles, Linux — /externalfiles), and relative paths. Relative paths are resolved against the root directory of Firebird. Given relative paths are not always obvious, it is recommended to use absolute paths.

This setting does not prevent creation of an external table with a disallowed path, but attempts to query the external table or insert into the external table will fail if the path is not allowed.

Attempts to access (select or insert) an external table file not allowed by the configuration will produce error “Use of external file at location <*filename*> is not allowed by server configuration” (error code `335544831` or `isc_conf_access_denied`).

For more information on external tables, see section External Tables in the *Firebird 5.0 Language Reference*.

Security recommendation

Never use `Full` for `ExternalFileAccess`, and be careful with the paths listed in `Restrict`.

External tables use a binary format, and a carefully crafted external table (or multiple external tables pointing to the same file) can be used to read any file, or append data to any file. For example, an external table with a `BINARY(1)` column can be used to read a file one byte per row, and append one byte to a file per insert. Such a method can be used to exfiltrate data from *any* file the server allows you to read, or append data to any file the server can write (either corrupting files, or adding data that didn’t previously exist).

Obviously, this requires that the user can create an external table in a database, and that the user running the Firebird server process can read and/or write these files, so there are limits to exploitability, but nevertheless, this is a scenario you need to consider when setting `ExternalFileAccess` to anything other than `None`.

For this reason we also recommend you ensure there is no overlap with the directories listed in `DatabaseAccess` or `UdfAccess`. That is, directories listed in `ExternalFileAccess` should not be the same as, be contained in, or contain the directories listed in `DatabaseAccess` or `UdfAccess`.

Overlap between the `DatabaseAccess` and `ExternalFileAccess` settings may allow an unprivileged user to copy database files by selecting from an external table pointing to the database file, or to create databases by inserting into an external table.

Overlap between the `UdfAccess` and `ExternalFileAccess` settings may allow a user to copy your UDF libraries (and for example look for vulnerabilities), or to add UDF libraries. The latter may compromise your system if this location is searched before another location for UDFs, loading a modified UDF library, or if the user has privileges to create UDFs in a database, they can now use this newly introduced UDF library.

Where possible, configure `ExternalFileAccess` per database, and ensure that databases cannot access each others external files, unless that is intended functionality (e.g. to share data between databases).

Examples

```
# No access
ExternalFileAccess = None
# Full access (DO NOT USE!)
ExternalFileAccess = Full
# Restricted (Windows)
ExternalFileAccess = Restrict C:\ExternalFiles;D:\Temp\ExternalFiles
# Restricted (Linux)
ExternalFilesAccess = /var/externalfiles;/tmp/externalfiles
```

See also

DatabaseAccess, UdfAccess

2.1.4. UdfAccess

Configures paths allowed for loading UDF — User-Defined Function — libraries.

Configuration

Global

Availability

Deprecated 4.0

Syntax

```
UdfAccess = <udf-access-config>

<udf-access-config> ::=
    None
  | Full
  | Restrict <path-list>

<path-list> ::= path [; path ...]
```

Default

None

Supported values

None

UDF libraries are not loaded.

Full

All paths are allowed for UDF libraries

Restrict <path-list>

Files with a path rooted in the directories listed in <path-list> (that is, the file is in a listed directory or in a subdirectory of a listed directory) are allowed for UDF libraries.

The <path-list> is a semicolon-separated list of directories. It is possible to use absolute paths (e.g. Windows — C:\UdfLibs, Linux — /udflibs), and relative paths. Relative paths are resolved against the root directory of Firebird. Given relative paths are not always obvious, it is recommended to use absolute paths.

UDFs and — by extension — UdfAccess are deprecated. The recommended replacements are built-in functions, PSQL functions (introduced in Firebird 3.0), or UDR functions (User-Defined Routine, introduced in Firebird 3.0).

Security recommendation

UDFs are inherently insecure and can compromise the security of your system and database. If possible, do not use UDFs and keep UdfAccess set to None, and if you do need UDFs, do not use Full, but explicitly list the allowed directories using Restrict.

We also recommend that there is no overlap with the directories listed in ExternalFileAccess. That is, directories listed in UdfAccess should not be the same as, be contained in, or contain the directories listed in ExternalFileAccess. See ExternalFileAccess for more information.

Examples

```
# No access
UdfAccess = None
# Full access (DO NOT USE!)
UdfAccess = Full
# Restricted (Windows)
UdfAccess = Restrict UDF;C:\UDF
# Restricted (Linux)
ExternalFilesAccess = UDF;/var/udf
```

See also

ExternalFileAccess

2.1.5. TempDirectories

Directories used by the Firebird engine for temporary files.

Configuration

Global

Syntax

```
TempDirectories = <path-list>

<path-list> ::= path [; path ...]
```

Default

Value of environment variable FIREBIRD_TMP (or its fallback).

The <path-list> is a semicolon-separated list of directories. It is possible to use absolute paths (e.g. Windows — C:\Database, Linux — /db), and relative paths. Relative paths are resolved against the root directory of Firebird. Given relative paths are not always obvious, it is recommended to use absolute paths.

Firebird will use the first directory listed until it runs out of disk space, then switch to the next directory in the list, and so on.

This setting does not affect the location of lock files.

Examples

```
# Multiple directories (Windows)
TempDirectories = C:\Temp\Firebird;D:\Temp\Firebird
# Single directory (Linux)
TempDirectores = /tmp/firebird
```

See also

TempTableDirectory, FIREBIRD_TMP

2.1.6. TempTableDirectory

Directory used for temporary tables and temporary blobs.

Configuration

Global, and per-database

Availability

Added 4.0

Syntax

```
TempTableDirectory = path
```

Default

Value of environment variable FIREBIRD_TMP (or its fallback).

Contrary to TempDirectories, this setting only accepts a single directory. If the configured directory does not exist or is not accessible, Firebird will use the directory specified in FIREBIRD_TMP or its fallback.

Examples

```
# Windows
TempTableDirectory = C:\Temp\Firebird
# Linux
```

```
TempTableDirectory = /tmp/firebird
```

See also

TempDirectories, FIREBIRD_TMP

2.1.7. AuditTraceConfigFile

Trace configuration file for system audit.

Configuration

Global

Syntax

```
AuditTraceConfigFile = path
```

Default

No value

The AuditTraceConfigFile sets the configuration file for system-wide auditing using the trace facility. System audit is only enabled if this configuration item is set to a valid trace configuration file.

It is possible to use absolute paths (e.g. Windows — C:\Firebird\fbtrace.conf, Linux — /fb/fbtrace.conf), and relative paths. Relative paths are resolved against the root directory of Firebird.

Examples

```
# Relative path
AuditTraceConfigFile = fbtrace.conf
# Absolute path (Windows)
AuditTraceConfigFile = C:\Firebird\customtrace.conf
# Absolute path (Linux)
AuditTraceConfigFile = /fb/customtrace.conf
```

See also

Trace configuration

2.1.8. MaxUserTraceLogSize

Maximum size in MiB of user trace session log files.

Configuration

Global

Syntax

```
MaxUserTraceLogSize = integer
```

Unit

Megabyte (MiB)

Default

`10` (10 MiB)

When the log file of a user trace session reaches the configured size, the trace session is suspended until the trace data is read through the trace service API (which clears/deletes the outstanding log file).

Example

```
MaxUserTraceLogSize = 15
```

2.1.9. DefaultDbCachePages

Default number of cached database pages

Configuration

Global, and per-database

Syntax

```
DefaultDbCachePages = integer
```

Unit

Database page

Default

SuperServer `2048` (per database)

SuperClassic `256` (per connection)

Classic `256` (per connection)

Firebird uses the page cache to store recently used database pages in memory. For SuperServer, the page cache is *per database*, and it's shared by all connections to that database; for SuperClassic and Classic, the cache is *per connection*.

On SuperServer, the cache for each database will use `(page count) * (page size)`.
On (Super)Classic, the cache will use `(page count) * (page size) * (connection count)`.

This setting only takes effect if the database has not been configured with an explicit number of page buffers using `gfix -buffers`. A backup with `gbak` will record the current configuration in the

backup file — even if it is derived from DefaultDbCachePages and not explicitly set. After a restore, the database will have this value explicitly configured in its database header, unless overridden with -BUFFERS.

On (Super)Classic the cache can also be overridden for a specific connection using the isc_dpb_num_buffers connection property.

Example

```
DefaultDbCachePages = 4096
```

2.1.10. DatabaseGrowthIncrement

Maximum database growth increment in bytes.

Configuration

Global, and per-database

Syntax

```
DatabaseGrowthIncrement = integer
```

Unit

Byte

Default

128M (128 MiB)

When a database has to allocate a database page, and there are no free pages, Firebird will allocate space with the following constraints:

- Minimum: 128 KiB
- 1/16th of existing allocated space
- Maximum: value of DatabaseGrowthIncrement

Preallocation reduces physical file fragmentation, and may improve performance.

If DatabaseGrowthIncrement is set to zero (0), preallocation is disabled, and Firebird will then allocate space per page.

Database shadow files are not preallocated.

Example

```
DatabaseGrowthIncrement = 512K
```

2.1.11. UseFileSystemCache

Configures if Firebird uses the filesystem cache for database files.

Configuration

Global, and per-database

Availability

Added 4.0

Syntax

```
UseFileSystemCache = Boolean
```

Default

`true`

This setting is ignored if it is not explicitly set, and `FileSystemCacheThreshold` has been explicitly set.

Example

```
# Disable filesystem cache
UseFileSystemCache = false
```

2.1.12. FileSystemCacheThreshold

Enables filesystem caching of a database if its page cache size is less than the configured value.

Configuration

Global, and per-database

Availability

Deprecated 4.0

Syntax

```
FileSystemCacheThreshold = integer
```

Unit

Database page

Default

`64K` (see also note below)

This setting is ignored if:

1. it has not been set explicitly, or
2. if `UseFileSystemCache` has also been set explicitly.

If set explicitly and UseFileSystemCache is not set explicitly, the filesystem cache is used if the page cache size configured in the database header — or if not set in the database header, the value of DefaultDbCachePages — is less than FileSystemCacheThreshold.

This setting has been deprecated in Firebird 4.0. The replacement is UseFileSystemCache.

Example

```
FileSystemCacheThreshold = 8192
```

2.1.13. FileSystemCacheSize

The maximum percentage of RAM used for the filesystem cache on 64-bit Windows.

Configuration

Global

Syntax

```
FileSystemCacheSize = integer
```

Unit

Percent

Range

0, 10 - 95

Default

0

This setting only has effect on 64-bit Windows; it is ignored on 32-bit Windows and other operating systems.

If 0, Firebird will use the current filesystem cache settings and will not attempt to change the setting.

Values in the range of 10 to 95 specify the maximum percentage of RAM the host will use for the filesystem cache. Values outside this range — other than 0 — will apply a default value of 30.

This setting configures the maximum size of the filesystem cache of Windows, and will not only affect Firebird.

The user running the firebird.exe server process needs to have the SeIncreaseQuotaPrivilege to change the maximum filesystem cache size. Built-in service accounts and administrators have it by default. If you use a custom user account for running Firebird, you will need to assign this privilege yourself. If the engine fails to adjust the cache size setting, it will log a warning message in

`firebird.log` and continue.

If it has already been set before Firebird was started, a restart may be required for the change to take effect.

2.1.14. RemoteFileOpenAbility

Configures if Firebird allows opening of database files on mounted network volumes.

Configuration

Global

Syntax

```
RemoteFileOpenAbility = Boolean
```

Default

`0` (`false`)

Enabling this option can cause database corruption

Do not enable this option unless you really know what you're doing!

This option removes an important safety feature of Firebird and can cause irrecoverable database corruption. Do not use this option unless you understand the risks and are prepared to accept the loss of the contents of your database.

Unless this configuration option is changed from `0` to 1, Firebird can only open a database if the database is stored on a drive physically attached to the local computer — the computer running that copy of Firebird. Requests for connections to databases stored on NFS mounted drives are redirected to a Firebird server running on the computer that "owns" the disk.

This restriction prevents two different copies of Firebird from opening the same database without coordinating their activities. Uncoordinated access by multiple copies of Firebird will corrupt a database. On a local system, the system-level file locking prevents uncoordinated access to the database file.

NFS does not provide a reliable way to detect multiple users of a file on an NFS mounted disk. If a second copy of Firebird connects to a database on an NFS mounted disk, it will corrupt the database. Under some circumstances, running a Firebird server on the computer that owns NFS mounted volumes is inconvenient or impossible. Applications that use the "embedded" variant of Firebird and never share access to a database can use this option to permit direct access to databases on NFS mounted volumes.

The situation for SMB/CIFS is quite similar to NFS, with not all configurations providing file locking mechanisms needed for safe operation. Using SuperServer with the database on a Windows Server file server may be considered relatively

> safe as file locking protects the database from being used by several engines. The network stack can still change order of writes, so you may get a corrupted database in case of network errors or power outage.
>
> The useful and safe case is working with a shared database marked read-only.

By default, Firebird only opens database files on local disks. When `RemoteFileOpenAbility` is enabled, Firebird can also open databases from mounted NFS volumes (Linux) or SMB/CIFS volumes (Windows)

Example

```
RemoteFileOpenAbility = 0
```

2.1.15. TempBlockSize

Allocation block size for temporary storage.

Configuration

Global

Syntax

```
TempBlockSize = integer
```

Unit

Byte

Default

`1M` (1 MiB)

Temporary storage is allocated in increments of `TempBlockSize`.

Example

```
TempBlockSize = 2M
```

2.1.16. TempCacheLimit

Maximum amount of temporary space that is cached in memory.

Configuration

Global, and per-database

Syntax

```
TempCacheLimit = integer
```

Unit

Byte

Default

SuperServer 64M (64 MiB)

SuperClassic 64M (64 MiB)

Classic 8M (8 MiB)

For Classic, the default is lower because the memory is allocated per server process, and this means the memory requirement grows with each connection.

Example

```
TempCacheLimit = 1G
```

2.1.17. MaxIdentifierByteLength

Maximum identifier length in bytes.

Configuration

Global, and per-database

Availability

Added 4.0

Syntax

```
MaxIdentifierByteLength = integer
```

Unit

Byte

Range

1 - 252

Default

252

Since Firebird 4.0, identifiers can be a maximum 252 bytes, or 63 characters UTF8. Set to 31 to use the same limit as in Firebird 3.0 and earlier.

This setting combined with MaxIdentifierCharLength determines the actual maximum length. For example, if MaxIdentifierCharLength setting is 31, and MaxIdentifierCharLength is not set, or set to 124 or higher, then an identifier can have maximum 31 characters of any UTF8 character. On the other hand, if MaxIdentifierCharLength is set to 31, and MaxIdentifierCharLength is also set to 31, then you can have 31 characters from the ASCII range, but using characters that require 2, 3, or 4 bytes to be encoded will reduce the actual maximum length in characters (to a minimum of 7 if all characters

are 4 bytes in UTF8).

Setting this value too low may make databases with longer identifiers inaccessible. For that reason, do not configure this globally, but only configure it per database in `databases.conf` when you *know* the identifiers will not be longer.

In general, we recommend not to touch this setting and use the default. Only use this for compatibility reasons, for example for applications that break when receiving longer identifiers that you cannot fix in any other way.

Example

```
MaxIdentifierByteLength = 31
```

See also

`MaxIdentifierCharLength`

2.1.18. MaxIdentifierCharLength

Maximum identifier length in characters.

Configuration

Global, and per-database

Availability

Added 4.0

Syntax

```
MaxIdentifierCharLength = integer
```

Unit

Character

Range

1 - 63

Default

63

Since Firebird 4.0, identifiers can be a maximum 252 bytes, or 63 characters UTF8. Set to 31 to use the same limit as in Firebird 3.0 and earlier.

This setting combined with `MaxIdentifierByteLength` determines the actual maximum length. For example, if `MaxIdentifierCharLength` setting is 31, and `MaxIdentifierCharLength` is not set, or set to 124 or higher, then an identifier can have maximum 31 characters of any UTF8 character. On the other hand, if `MaxIdentifierCharLength` is set to 31, and `MaxIdentifierCharLength` is also set to 31, then you can have 31 characters from the ASCII range, but using characters that require 2, 3, or 4 bytes to be

encoded will reduce the actual maximum length in characters (to a minimum of 7 if all characters of an identifier are 4 bytes in UTF8).

Setting this value too low may make databases with longer identifiers inaccessible. For that reason, do not configure this globally, but only configure it per database in `databases.conf` when you *know* the identifiers will not be longer.

In general, we recommend not to touch this setting and use the default. Only use this for compatibility reasons, for example for applications that break when receiving longer identifiers that you cannot fix in any other way.

Example

```
MaxIdentifierCharLength = 31
```

See also

`MaxIdentifierByteLength`

2.1.19. InlineSortThreshold

Maximum sort record size that can be stored inline in the sort block.

Configuration

Global, and per-database

Syntax

```
InlineSortThreshold = integer
```

Unit

Byte

Default

1000

This controls if non-key fields are stored inside the sort block, or to refetch them from the data pages after sorting. If the sort record size with non-key fields exceeds the configured maximum, only the key fields are stored in the sort block.

A value of `0` (zero) disables storing non-key fields and always refetches non-key fields.

This setting is a balance between the cost of refetching data, and the cost of sorting when including non-key data. If a sort block is very large (e.g. due to a lot of rows and/or large sort record size), the sort will be performed on disk. The I/O cost of sorting including non-key fields may be larger than the I/O cost of refetching data.

Examples

```
# 2000 bytes
```

```
InlineSortThreshold = 2000

# Disabled
InlineSortThreshold = 0
```

2.1.20. OptimizeForFirstRows

Enables first row optimization strategy.

Configuration

Global, and per-database

Availability

Added 5.0

Syntax

```
OptimizeForFirstRows = Boolean
```

Default

`false` (optimize for all rows)

Defines whether queries should be optimized to return the first row(s) as soon as possible rather than returning the whole dataset as soon as possible. By default, Firebird optimize for retrieval of all rows.

This can be overridden at the session level using the `SET OPTIMIZE` statement, or at the statement level by using the `OPTIMIZE FOR` clause.

Example

```
OptimizeForFirstRows = true
```

2.1.21. OuterJoinConversion

Enables optimizer conversion of outer joins to inner joins.

Configuration

Global, and per-database

Availability

Added 5.0

Deprecated 5.0

Syntax

```
OuterJoinConversion = Boolean
```

Default

`true` (enabled)

Defines whether the optimizer attempts to convert outer joins into inner joins, provided that such a transformation is possible from the query result perspective. This can be done if — for example — a `LEFT OUTER JOIN` is used, but the "missing" rows of the right table are filtered out in the `WHERE` by a condition that excludes `NULL` for a column of that table.

Enabled by default. Disable to simplify the migration path if outer joins are used intentionally in SQL queries (e.g. as optimizer hints) even if they are known to be semantically equivalent to inner joins.

This setting is intended as a temporary backward compatibility option. If you have a reason to disable this, we'd appreciate us if you tell us why on firebird-devel.

There is no guarantee that this setting will be available in future Firebird versions.

Example

```
# Disable outer join conversion
OuterJoinConversion = false
```

2.1.22. SubQueryConversion

Enables experimental optimizer conversion of subqueries in `EXISTS` and `IN` to semi-joins.

Configuration

Global, and per-database

Availability

Added	5.0
Deprecated	5.0

Syntax

```
SubQueryConversion = Boolean
```

Default

`false` (disabled)

Defines whether the optimizer attempts to merge subqueries in `EXISTS` and `IN` with the outer query by converting them to semi-joins, provided that such a transformation is possible from the query result perspective.

This is an experimental feature in Firebird 5.0, so it's disabled by default. Enable and give it a try for a possibly improved performance due to subqueries being evaluated just once and then cached.

If you run into problems like decreased performance or wrong query results with this option enabled, we'd appreciate us if you tell us on firebird-devel or on the issue tracker.

Once this feature is proved to be superior in most use cases, this setting will be removed, and this conversion will become unconditional.

There is no guarantee that this setting will be available in future Firebird versions.

Example

```
SubQueryConversion = true
```

2.1.23. AuthServer

Authentication plugins accepted by Firebird server.

Configuration

Global, and per-database

Syntax

```
AuthServer = <plugin-list>

<plugin-list> ::= plugin [<separator> plugin ...]

<separator> ::=
  one of <space> (' '), <comma> (','), <semicolon> (';')
```

Default

`Srp256`

Authentication plugins have a server-side part and client-side part. The authentication plugins implement how a user is authenticated. Some authentication plugins also generate the encryption key for wire encryption. Generally, users are managed by a user manager that is specific to a plugin or family of plugins.

The authentication plugins the client tries are specified by the `AuthClient` setting.

A default Firebird installation supports the following authentication plugins:

Standard authentication plugins included in Firebird

Srp	SRP (Secure Remote Password) with SHA-1 client-proof
Srp224	SRP with SHA-224 client-proof

`Srp256`	SRP with SHA-256 client-proof
`Srp384`	SRP with SHA-384 client-proof
`Srp512`	SRP with SHA-512 client-proof
`Win_Sspi`	Windows SSPI authentication (a.k.a. trusted authentication); Windows-only. Users are not managed in Firebird, but may require a mapping (`CREATE [GLOBAL] MAPPING`), e.g. for role assignment based on Windows groups, etc.
`Legacy_Auth`	Legacy authentication (max 8 character password; considered insecure!)

From these plugins, only `Legacy_Auth` does not generate an encryption key for wire encryption. Additional third-party plugins may be available.

Secure Remote Password and hashes

Firebird's SRP implementation internally uses SHA-1 for hashing. The various Srp*NNN* plugins only change the hash used for the client-proof.

See also firebird#6051.

Example

```
AuthServer = Srp256, Srp
```

See also

`AuthClient`, `UserManager`

2.1.24. AuthClient

Authentication plugins tried by Firebird *fbclient*.

Configuration

Global, per-database, client-side, and per-connection

Syntax

```
AuthClient = <plugin-list>

<plugin-list> ::= plugin [<separator> plugin ...]

<separator> ::=
  one of <space> (' '), <comma> (','), <semicolon> (';')
```

Default

`Srp256, Srp, Legacy_Auth` (non-Windows)
`Srp256, Srp, Win_Sspi, Legacy_Auth` (Windows)

The authentication plugins determine how a user authenticates. Some authentication plugins also generate the encryption key for wire encryption.

For a list of plugins supported by a default Firebird installation, see Standard authentication plugins included in Firebird.

When configured server-side (in `firebird.conf` or `databases.conf`), this setting controls which plugins are tried when the server creates a connection to another database (e.g. using `EXECUTE STATEMENT ⋯ ON EXTERNAL`).

When configured client-side — in `firebird.conf` read by the client, or using `isc_dpb_config` (key/value) or `isc_dpb_auth_plugin_list` (value only) — it controls which plugins are tried for connecting.

The server and client must agree on the plugins used. The list of plugins accepted by the server is configured through `AuthServer`. However, the client will always try the first plugin listed, even if not supported by the server. Subsequent plugins will only be tried if they are also listed in the `AuthServer` setting of the server.

Insecurity of `Legacy_Auth`

The `Legacy_Auth` plugin is insecure, as it sends a UnixCrypt hash of the first 8 bytes of the password over the wire. This hash is easily crackable with a rainbow table, and the method of transfer is susceptible to replay attacks. In general, it is advisable to remove the `Legacy_Auth` plugin from consideration entirely.

If `Legacy_Auth` is listed first in `AuthClient`, it will send an easily crackable hash of the first 8 bytes of the password, even if the user in question is — for example — an SRP user. This also happens if `Legacy_Auth` is listed in both `AuthClient` (of the client) and `AuthServer` (of the server), and none of the earlier plugins authenticated the user.

Example

```
AuthClient = Srp256
```

See also

`AuthServer`

2.1.25. UserManager

Enabled user manager plugins.

Configuration

Global, and per-database

Syntax

```
UserManager = <plugin-list>
```

```
<plugin-list> ::= plugin [<separator> plugin ...]

<separator> ::=
  one of <space> (' '), <comma> (','), <semicolon> (';')
```

Default

`Srp`

User manager plugins manage users, generally in the security database. The user manager plugins are also used to populate the `SEC$USERS` and `SEC$USER_ATTRIBUTES` virtual tables.

The first plugin listed is the *default* user manager. The default user manager is used if the `USING PLUGIN` clause is absent from user management statements. The default user manager is also used by *gsec* and the user management service API[4].

Setting `UserManager` explicitly to an empty value (blank) will disallow management of users.

A default Firebird installation supports the following user manager plugins:

Standard user manager plugins included in Firebird

`Srp`	User manager for the SRP-family of authentication plugins
`Legacy_UserManager`	User manager of the `Legacy_Auth` authentication plugin

The per-database configuration in `databases.conf` enables which user manager plugins are allowed — and which is the default — when connecting to that specific database. When configured on a security database, it only manages which user manager plugins are allowed when connecting directly to that security database; it does not configure which user manager plugins may store data in that security database.

Security recommendations

1. Where possible, replace insecure `Legacy_Auth`/`Legacy_UserManager` users with `Srp` users (and drop those insecure users).
2. Some Firebird documentation suggest that if you need to manage `Legacy_Auth` users, to put `Legacy_UserManager` first. **Do not do that unless you absolutely have to!**

   Listing `Legacy_UserManager` first means that you will create insecure users by default if you forget to specify the `USING PLUGIN` clause, or use the deprecated *gsec* or user management service API. Creating insecure users should be a conscious and explicit choice (i.e. by specification of `USING PLUGIN Legacy_UserManager` in the statement).

   You may even want to consider not listing `Legacy_UserManager`, except on a specific database to manage these insecure users.
3. Consider setting `UserManager` explicitly blank in `firebird.conf`, and selectively enable user management for specific databases in `databases.conf`. If you enable this only for a database marked with `RemoteAccess = false` (like the default

security database), you will also disallow remote user management.

Doing this may improve security, by preventing attack vectors that exploit loopholes/bugs or insecure/leaked admin accounts to drop, create, or alter users, but it will also make user management harder. Disabling user management this way will also prevent users from changing their own passwords (e.g. using ALTER CURRENT USER).

Examples

```
# Enable Srp and Legacy_UserManager, with Srp as default
UserManager = Srp, Legacy_UserManager
# Disable user management
UserManager =
```

See also

AuthServer

2.1.26. DefaultProfilerPlugin

Default profiler plugin

Configuration

Global, and per-database

Availability

Added 5.0

Syntax

```
DefaultProfilerPlugin = plugin
```

Default

Default_Profiler

Specifies the default profiler plugin used by the RDB$PROFILER package, if no explicit plugin name is specified (parameter PLUGIN_NAME of START_SESSION).

Example

```
# Set default to MyCustomProfiler
# This requires a custom plugin-library with that name
DefaultProfilerPlugin = MyCustomProfiler
```

2.1.27. TracePlugin

The trace plugin used by the Firebird trace facility.

Configuration

Global

Syntax

```
TracePlugin = plugin
```

Default

`fbtrace`

The Firebird trace facility can be used to audit or record statements executed on the server. The trace plugin is responsible for writing the data to a log file in a specific format.

Firebird includes a default trace plugin called `fbtrace`. There are third-party plugins available which provide different output formats.

Example

```
TracePlugin = customtrace
```

2.1.28. WireCryptPlugin

Wire crypt plugins tried by the client and/or accepted by the server.

Configuration

Global, per-database, client-side, and per-connection

Syntax

```
WireCryptPlugin = <plugin-list>

<plugin-list> ::= plugin [<separator> plugin ...]

<separator> ::=
  one of <space> (' '), <comma> (','), <semicolon> (';')
```

Default

`ChaCha64, ChaCha, Arc4`

Wire crypt plugins allow for encryption of the database connection *after* authentication. The default plugins included in Firebird use a session-key generated by the authentication plugin[5] to encrypt the connection.

A default Firebird installation supports the following wire encryption plugins:

Standard wire encryption plugins included in Firebird

ChaCha ChaCha#20 (RFC 8439) with 32-bit counter (4 byte counter, 12 byte nonce, 20 (10 + 10) rounds). The session key is stretched (or reduced) to 256-bit using SHA-256. Session keys shorter than 16 bytes are rejected.

Due to implementation limitations, this plugin is limited to 256 GiB of data in one direction.

ChaCha64 ChaCha#20 variant with 64-bit counter (8 byte counter, 8 byte nonce, 20 (10 + 10) rounds). The session key is stretched (or reduced) to 256-bit using SHA-256. Session keys shorter than 16 bytes are rejected.

Arc4 Alleged RC4.

RC4 is considered insecure these days.

This configuration property serves multiple purposes:

1. server-side (`firebird.conf` or `databases.conf` of the server)
   - The plugins accepted for incoming connections
   - The plugins tried for outgoing connections (i.e. server as client)
2. client-side (`firebird.conf` read by client, or key/value in `isc_dpb_config`)
   - The plugins tried for outgoing connections

The intersection of server-side and client-side wire encryption plugins decides which plugins are tried. The server-side order of plugins decides in which order they are tried.

Example

```
# Only enable Arc4
WireCryptPlugin = Arc4
```

2.1.29. KeyHolderPlugin

Database encryption keyholder plugin

Configuration

Global, and per-database

Syntax

```
KeyHolderPlugin = plugin
```

Default

No value

A keyholder serves as a temporary storage for database encryption keys. Generally, this plugin will be provided by the third-party database encryption plugin.

Firebird does not include a default plugin. The Linux distribution includes an example in `libCryptKeyHolder_example.so`, but this library is intended as an example only, and is not for production use.

Example

```
KeyHolderPlugin = CustomKeyHolder
```

2.1.30. AllowEncryptedSecurityDatabase

Enables use of an encrypted security database.

Configuration

Global, and per-database

Syntax

```
AllowEncryptedSecurityDatabase = Boolean
```

Default

`false`

With the default setting, it is not possible to authenticate against an encrypted security database.

Depending on the database encryption plugin used and its configuration, the client may hold the encryption key and needs to send it to the server to be able to access the database. With the default configuration, this key exchange will only happen after authentication completed successfully and — if enabled — wire encryption was set up (which depends on the session key of authentication).

However, if the security database is encrypted, this key exchange must occur *before* authentication. As this needs to happen before wire encryption is established, and could lead to key exposure[6], this must be explicitly enabled.

In general, the security database — assuming use of SRP and long/quality passwords — is pretty good at protecting itself and doesn't need to be encrypted. One use case of an encrypted security database is if a database is its own security database and needs to be encrypted for other reasons.

Please understand what you are doing before enabling this feature.

Enabling this feature may expose the database encryption key if it is transmitted from client to server, as the encrypted security database needs to be accessed *before* wire encryption can be established, as that depends on the successful completion of authentication.

Enabling this option may also cause the database encryption handshake to occur before authentication and wire encryption even if the security database itself is not encrypted, but the destination database is.

Authenticating against an encrypted security database is not supported by the `Legacy_Auth` authentication plugin.

Example

```
AllowEncryptedSecurityDatabase = false
```

2.1.31. Providers

Connection providers used to connect to a database.

Configuration

Global, per-database, client-side, and per-connection

Syntax

```
Providers = <plugin-list>

<plugin-list> ::= plugin [<separator> plugin ...]

<separator> ::=
  one of <space> (' '), <comma> (','), <semicolon> (';')
```

Default

`Remote,EngineNN,Loopback` (with *NN* the major ODS of the Firebird version)

Providers are plugins used to connect to a database.

A default Firebird installation supports the following provider plugins:

Standard provider plugins included in Firebird

Remote Connect to a remote Firebird server using TCP/IP, or WNET (Windows-only, removed in Firebird 5.0).

This provider expects the legacy URL including a hostname (e.g. TCP/IP `servername/port:database`, or WNET `\\servername:port\database`), or explicit protocols (e.g.`xnet://database`, `inet://servername:port/database`, etc.)

EngineNN Connect using the Firebird ODS *NN* database engine (e.g. `Engine13` for Firebird 4.0 (ODS 13.0) or Firebird 5.0 (ODS 13.0 *and* ODS 13.1). Client-side, this is what is called an *embedded* connection.

This provider only expects a database name (filename or alias).

Loopback Connect to the locally running server. On Windows, it tries XNET, TCP/IP (using `localhost`) and finally WNET (prepending `\\.\`). On other OSes, only TCP/IP is tried.

The `Loopback` provider is basically the same as `Remote`, except it always connects to the local server.

This provider only expects a database name (filename or alias).

Additional providers may be added. For example, copying the Engine12.dll/libEngine12.so from Firebird 3.0 to the plugins directory of Firebird 4.0 or 5.0 and including Engine12 in Providers adds support for opening Firebird 3.0 (ODS 12) databases.

This configuration property serves multiple purposes:

1. server-side (firebird.conf or databases.conf of the server)
    - The plugins accepted for incoming connections. However, unless Redirection is enabled, Remote is ignored for incoming connections.
    - The plugins tried for outgoing connections (i.e. server as client)
2. client-side (firebird.conf read by client, or key/value in isc_dpb_config)
    - The plugins tried for outgoing connections

      For example, specifying only Engine13 will ensure the client will only make embedded connections.

Example

```
Providers = Engine13
```

2.1.32. DeadlockTimeout

Timeout for detecting and purging locks held by dead processes in case of a lock conflict.

Configuration

Global, and per-database

Syntax

```
DeadlockTimeout = integer
```

Unit

Second

Default

10

The deadlock timeout is the number of seconds the lock manager will wait in case of a lock conflict before purging locks of dead processes and doing an extra deadlock scan cycle. In normal cases, Firebird will detect deadlocks instantly, so this timeout is only to detect exceptional cases.

Setting this setting too low may increase load and reduce performance.

Example

```
DeadlockTimeout = 15
```

2.1.33. StatementTimeout

Statement timeout.

Configuration

Global, and per-database

Availability

Added 4.0

Syntax

```
StatementTimeout = integer
```

Unit

Second

Default

`0` (no timeout)

The statement timeout is the maximum duration of statement execution and cursor use. The engine will automatically cancel the statement after the configured number of seconds. A value of `0` means that no database-level statement timeout is set.

The statement timeout can be configured on three levels:

1. Database level (through `firebird.conf` or `databases.conf`)
2. Connection level (through `SET STATEMENT TIMEOUT` or the API)

   If the database-level timeout is non-zero and lower than the connection-level timeout, the database-level timeout applies instead.

3. Statement level (through the API)

   If the database-level timeout is non-zero and lower than the statement-level timeout, the database-level timeout applies instead. A non-zero statement-level timeout takes precedence over the connection-level timeout (i.e. the connection-level timeout is not an upper bound).

The timeout is reported when the client sends a request involving the statement handle. The request will complete with error `isc_cancelled` and a secondary error code indicating which timeout expired:

`isc_cfg_stmt_timeout`	Database-level timeout expired
`isc_att_stmt_timeout`	Connection-level timeout expired
`isc_req_stmt_timeout`	Statement-level timeout expired

Timeout includes duration of open cursor

In the case of a result set producing statement (e.g. a SELECT), the timeout covers the time from statement execution until cursor close. That means that even if Firebird executes the query quickly, but the application takes a long time to fetch rows from the cursor (e.g. because it waits for user action — like scrolling a grid — to fetch rows), the statement will be cancelled if the fetch occurs after the timeout.

In other words, be careful with setting this too low.

Notes about statement timeouts

1. A client application could wait longer than the time set by the timeout value if the engine needs to undo a large number of actions as a result of the statement cancellation
2. When the engine runs an EXECUTE STATEMENT statement, it passes the remainder of the currently active timeout to the new statement. If the external (remote) engine does not support statement timeouts, the local engine silently ignores any corresponding error.
3. When the engine acquires a lock from the lock manager, it tries to lower the value of the lock timeout using the remainder of the currently active statement timeout, if possible. Due to lock manager internals, any statement timeout remainder will be rounded up to whole seconds.

Example

```
# Five minute timeout
StatementTimeout = 300
```

2.1.34. ConnectionIdleTimeout

Connection idle timeout.

Configuration

Global, and per-database

Availability

Added 4.0

Syntax

```
ConnectionIdleTimeout = integer
```

Unit

Minute

Default

0 (no timeout)

The connection idle timeout — or session idle timeout — is the maximum time a connection may be idle (i.e. the engine receives no requests) before it shuts down the connection. A setting of `0` means that no database-level connection idle timeout is set.

By default, the idle timeout is not enabled. No minimum or maximum limit is imposed, but a reasonably large period — such as a few hours — is recommended.

The connection idle timeout can be configured on three levels:

1. Database level (through `firebird.conf` or `databases.conf`)
2. Connection level (through `SET SESSION IDLE TIMEOUT` or the API)

   If the database-level timeout is non-zero and lower than the connection-level timeout, the database-level timeout applies instead.

When a connection is shutdown by the expiration of the idle timeout, the next user API call returns the error `isc_att_shutdown` with secondary error `isc_att_shut_idle`.

Example

```
# One hour
ConnectionIdleTimeout = 60
```

2.1.35. OnDisconnectTriggerTimeout

Timeout of `ON DISCONNECT` triggers.

Configuration

Global, and per-database

Availability

Added 4.0.2

Syntax

```
OnDisconnectTriggerTimeout = integer
```

Unit

Second

Default

`180` (3 minutes)

Firebird will automatically cancel `ON DISCONNECT` triggers if they run longer than the configured timeout. A value of `0` means that no timeout is set.

Example

```
# Five minutes
```

```
OnDisconnectTriggerTimeout = 300
```

2.1.36. MaxUnflushedWrites

Maximum number of pending asynchronous writes.

Configuration

Global, and per-database

Syntax

```
MaxUnflushedWrites = integer
```

Unit

Data page/pending write

Default

100 (Windows)
-1 (other platforms)

This parameter determines how frequently pending asynchronous writes are flushed to disk when Forced Writes are disabled (or, asynchronous writing is enabled). Its value is the number of asynchronous writes that may be pending before a flush is flagged to be done next time a transaction commits. A setting of -1 disables this (i.e. pending writes are not flushed).

This setting only applies when forced-writes are off.

If the MaxUnflushedWriteTime expires before the maximum number of pending pages is reached, the currently pending pages are flushed as well.

Primarily for Windows

This setting was introduced in Firebird 1.5 to address problems on Windows where asynchronous writes were not written to disk until controlled shutdown of Firebird. If Firebird crashed, this could result in data loss.

Example

```
MaxUnflushedWrites = 50
```

See also

MaxUnflushedWriteTime

2.1.37. MaxUnflushedWriteTime

Maximum time asynchronous writes may be pending.

Configuration

Global, and per-database

Syntax

```
MaxUnflushedWriteTime = integer
```

Unit

Second

Default

5 (Windows)
-1 (other platforms)

This parameter determines how frequently pending asynchronous writes are flushed to disk when Forced Writes are disabled (or, asynchronous writing is enabled). Its value is the maximum time in seconds an asynchronous write may be pending before a flush is flagged to be done next time a transaction commits. A setting of -1 disables this (i.e. pending writes are not flushed).

This setting only applies when forced-writes are off.

If the MaxUnflushedWrites is reached before this timeout, the currently pending pages are flushed as well.

Primarily for Windows

This setting was introduced in Firebird 1.5 to address problems on Windows where asynchronous writes were not written to disk until controlled shutdown of Firebird. If Firebird crashed, this could result in data loss.

Example

```
# 10 seconds
MaxUnflushedWriteTime = 10
```

See also

MaxUnflushedWrites

2.1.38. BugcheckAbort

Enables bugcheck abort.

Configuration

Global

Syntax

```
BugcheckAbort = Boolean
```

Default

0 (false) — normal builds
1 (true) — debug builds (with DEV_DEBUG)

When enabled, Firebird calls abort() when internal errors or BUGCHECK macros are encountered. This invokes the post-mortem debugger which can create a core-dump for off-line analysis. When disabled, the engine tries to minimize damage and continue execution.

Setting this option to 1 will produce traceable core-dumps when something nasty like SIGSEGV happens inside a UDF. On Windows, enabling this option will invoke the JIT debugger facility when errors happen.

Example

```
# Enable bugcheck abort
BugcheckAbort = 1
```

2.1.39. ClearGTTAtRetaining

Configures if Global Temporary Tables (GTT) with ON COMMIT DELETE ROWS are cleared on commit/rollback retaining.

Configuration

Global, and per-database

Availability

Deprecated	3.0
Removed	5.0

Syntax

```
ClearGTTAtRetaining = Boolean
```

Default

0 (false)

This is a compatibility setting to restore behaviour of Global Temporary Tables as it was in Firebird 2.1. In the normal behaviour (ClearGTTAtRetaining = 0), a Global Temporary Table with ON COMMIT DELETE ROWS is cleared at a hard commit or rollback, but is retained and the data is accessible after a commit retain or rollback retain. When enabled (ClearGTTAtRetaining = 1), the table is also cleared on commit retain or rollback retain.

This is intended as an interim workaround to be able to run legacy code which relies on the old behaviour. We recommend that you fix such code, so you don't rely on this workaround.

This setting was removed in Firebird 5.0.

Example

```
# Enable clearing GTT at commit/rollback retain
ClearGTTAtRetaining = 1
```

2.1.40. RelaxedAliasChecking

Relaxes relation alias checking rules in SQL.

Configuration

Global

Availability

Deprecated 2.0

Syntax

```
RelaxedAliasChecking = Boolean
```

Default

`0` (false)

In Firebird 2.0, stricter parser rules for aliases of relations (e.g. tables, views) were implemented to conform to SQL standard requirements. Enabling this setting relaxes the relation alias checking rules to the Firebird 1.5 and older behaviour.

With the default setting, if a relation is aliased, you are only allowed to reference that relation using its alias. If `RelaxedAliasChecking` is set to `1` (true), you can also reference the relation using its original name.

For example:

```
-- Reference by alias (strict and relaxed)
select E.FIRST_NAME
from EMPLOYEE as E

-- Reference by relation name (relaxed only)
select EMPLOYEE.FIRST_NAME
from EMPLOYEE as E
```

This is intended as an interim workaround to be able to run legacy code which relies on the old behaviour. We recommend that you fix such code, so you don't rely on this workaround.

There is no guarantee that this setting will be available in future Firebird versions.

Example

```
# Relaxed (non-strict) alias checking
RelaxedAliasChecking = 1
```

2.1.41. ReadConsistency

Configures if READ COMMITTED transactions always use READ CONSISTENCY mode.

Configuration

Global, and per-database

Availability

Added 4.0

Syntax

```
ReadConsistency = Boolean
```

Default

1 (true)

Since Firebird 4.0, READ COMMITTED by default always uses the READ CONSISTENCY mode, even if [NO] RECORD VERSION (isc_tpb_rec_version/isc_tpb_no_rec_version in TPB) is requested. If ReadConsistency is set to 0 (false), the requested mode is used.

When disabled, READ CONSISTENCY mode must be requested explicitly (e.g. SET TRANSACTION READ COMMITTED READ CONSISTENCY or by including isc_tpb_read_consistency in the TPB).

This setting is intended for backwards compatibility, if your application specifically depends on the behaviour of [NO] RECORD VERSION. In general, we recommend to leave it at its default value.

This is intended as an interim workaround to be able to run legacy code which relies on the old behaviour. We recommend that you fix such code, so you don't rely on this workaround.

There is no guarantee that this setting will be available in future Firebird versions.

Example

```
# Disable always using read consistency mode for READ COMMITTED
ReadConsistency = 0
```

2.1.42. DataTypeCompatibility

Configures if newer data types are automatically converted to data types available in an older version.

Configuration

Global, and per-database

Availability

Added 4.0

Syntax

```
DataTypeCompatibility = [{ 2.5 | 3.0 }]
```

Default

No value

The DataTypeCompatibility setting controls if input parameters (e.g. query parameters) and output columns (e.g. columns of a result set) using new data types are automatically converted to "legacy" data types of the specified older version.

Setting to an empty value will return the data type as is. This can be used in databases.conf to override (clear) the configuration from firebird.conf.

Configure this setting as a last resort, as it affects all clients to the server — if set in firebird.conf — or database — in databases.conf — even if a client supports the newer data types. Whenever possible, use the isc_dpb_set_bind connection property or the SET BIND statement instead.

Table 1. DataTypeCompatibility *coercion rules*

Source data type	Target data type	Active for
BOOLEAN	CHAR(5)	2.5
DECFLOAT	DOUBLE PRECISION	2.5, 3.0
INT128	BIGINT	2.5, 3.0
NUMERIC(p[,s]) p > 18	NUMERIC(18[,s])	2.5, 3.0
DECIMAL(p[,s]) p > 18	DECIMAL(18[,s])	2.5, 3.0
TIME WITH TIME ZONE	TIME WITHOUT TIME ZONE	2.5, 3.0
TIMESTAMP WITH TIME ZONE	TIMESTAMP WITHOUT TIME ZONE	2.5, 3.0

This setting only affects how statement bindings of input parameters and output columns are described to the client, and how values of those parameters and columns are converted between the client and server. It does not change the data types reported by the metadata tables, or on the server itself.

Example

```
DataTypeCompatibility = 2.5
```

2.1.43. ConnectionTimeout

Connection timeout.

Configuration

Global, client-side, and per-connection

Syntax

```
ConnectionTimeout = integer
```

Unit

Second

Default

180 (3 minutes)

The ConnectionTimeout is the number of seconds the client (or the server acting as client to another database) waits before concluding the connection attempt failed.

Example

```
# Connection timeout of 5 minutes
ConnectionTimeout = 300
```

2.1.44. WireCrypt

Controls if wire encryption is used or even required.

Configuration

Global, per-database, client-side, and per-connection

Syntax

```
WireCrypt = { Disabled | Enabled | Required }
```

Default

Required (server)
Enabled (client)

By default, encryption is *Required* for incoming connections and *Enabled* for outgoing connections (client connections).

To access a server using an older client library which does not support wire encryption, WireCrypt in the server configuration file should be set to Enabled instead of the default Required. It is recommended to not set WireCrypt to Disabled, because then newer clients will also not encrypt their connections.

The rules are:

- If one side has `WireCrypt = Disabled`:
  - If the other side has `WireCrypt = Required`, the connection is rejected
  - Otherwise, the connection is established without encryption
- If the server and client don't have matching wire encryption plugins, or the encryption key is missing:
  - If either side has `WireCrypt = Required`, the connection is rejected
  - Otherwise, the connection is established without encryption
- In all other cases, an encrypted connection is established.

Example

```
WireCrypt = Enabled
```

2.1.45. WireCompression

Enables (zlib) wire compression of connections.

Configuration

Client-side, and per-connection

Syntax

```
WireCompression = Boolean
```

Default

`false`

This setting is ignored if the client library does not have access to the zlib library (`zlib1.dll` /`libz.so.1`).

It is not possible to disable wire compression server-side; if the client requests compression, the server will enable compression.

Example

```
# Enable wire compression
WireCompression = true
```

2.1.46. DummyPacketInterval

Seconds to wait on a silent client connection before the server sends a dummy packet to request acknowledgment.

Configuration

Global, per-database, client-side, and per-connection

Syntax

```
DummyPacketInterval = integer
```

Unit

Second

Default

`0` (disabled)

The dummy packet interval is a way to detect broken connections. If a client is idle for more than the configured interval, the server will send a dummy packet to the client. This packet will result in a TCP/IP-level acknowledgement if the connection is still alive. Otherwise, it will result in detection of the broken connection.

Do not set this setting too low: use a value of tens of minutes or even hours, not just a few seconds.

A too low value can result in a lot of unnecessary busywork for the server for little gain.

Configuring this setting should generally be unnecessary as Firebird configures its sockets with `SO_KEEPALIVE` which will do a similar check at a lower level. If you do not like the default 2-hour keepalive timeout used by the TCP/IP stack, adjust your server OS settings appropriately.

On UNIX-like OS's, modify contents of `/proc/sys/net/ipv4/tcp_keepalive_*` (which also apply to IPv6!).

For Windows, see the `KeepAliveTime` registry documentation.

Example

```
# Set interval to 30 minutes
DummyPacketInterval = 1800
```

2.1.47. `ClientBatchBuffer`

Buffer size (in bytes) used by the client connection to accumulate output messages before sending them to the server using the Batch API.

Configuration

Client-side, and per-connection

Availability

Added 4.0

Syntax

```
ClientBatchBuffer = integer
```

Unit

Byte

Default

131072 (or 128K)

This configuration item governs size of the client-side buffers for statement parameters *and* blobs for batch execution. Given blob and parameters are stored in separate buffers, the total space allocated is upto twice the configured size, per batch. Irrespective of the configured buffer size, the client will allocate sufficient space to buffer at least one set of statement parameters.

Example

```
# Buffer two megabyte
ClientBatchBuffer = 2M
```

2.1.48. DefaultTimeZone

Default session or client time zone.

Configuration

Global, client-side, and per-connection

Availability

Added 4.0

Syntax

```
DefaultTimeZone = time-zone-name
```

Default

No value (for server: derive from OS time zone)

- If empty, the default time zone for the server is derived from the OS time zone.

  The derived time zone uses the rules in the ICU time zone database loaded by Firebird, not the OS time zone database. Additional OS settings, like disabling daylight saving time, have no effect.

- When set in the server, it defines the default session time zone for attachments that don't specify isc_dpb_session_time_zone.
- When set in the client, it defines the default time zone used with client-side API functions and the default value of isc_dpb_session_time_zone.

The value of *time-zone-name* is a time zone identifier string, as documented in Time Zone Format in the *Firebird 5.0 Language Reference*.

Example

```
# Default to Europe/Berlin
DefaultTimeZone = Europe/Berlin
```

2.1.49. RemoteServiceName

The TCP service name/port number to be used for database connections.

Configuration

Global, client-side, and per-connection

Syntax

```
RemoteServiceName = service-name
```

Default

`gds_db`

If *service-name* is defined in the `services` definition file (e.g. `/etc/services` or `C:\Windows\System32\drivers\etc\services`), it is used to determine the TCP port to use. If *service-name* is not found, the value or default of `RemoteServicePort` is used.

Server-side

Determines the port that the server listens for connections

Client-side and per-connection

Determines the port to use if none is specified in a TCP/IP connection URL

The order of precedence is explicitly set values before defaults, and `RemoteServiceName` — if an entry is found in the `services` file — before `RemoteServicePort`. In other words, if `RemoteServiceName` is not explicitly set, and `RemoteServicePort` is explicitly set, the value of `RemoteServicePort` is used.

Example

```
RemoteServiceName = gds_db
```

See also

`RemoteServicePort`

2.1.50. RemoteServicePort

The TCP service port number to be used for database connections.

Configuration

Global, client-side, and per-connection

Syntax

```
RemoteServicePort = integer
```

Range

1 - 65535

Default

3050

Configures the TCP port to use.

Server-side

Determines the port that the server listens for connections

Client-side and per-connection

Determines the port to use if none is specified in a TCP/IP connection URL

The order of precedence is explicitly set values before defaults, and RemoteServiceName — if an entry is found in the services file — before RemoteServicePort. In other words, if RemoteServiceName is not explicitly set, and RemoteServicePort is explicitly set, the value of RemoteServicePort is used.

Example

```
# Use port 3051 instead of 3050
RemoteServicePort = 3051
```

See also

RemoteServiceName

2.1.51. RemoteAuxPort

The TCP port number to be used for server event notification messages.

Configuration

Global, and per-database

Syntax

```
RemoteAuxPort = integer
```

Range

0 - 65535

Default

0 (use random port)

When a client listens for events, it establishes a secondary connection to the Firebird server, using a port number the server sends to the client. By default, a random port is used.

This setting can be used to specify a fixed port number, for example one that can be allowed through the firewall.

Example

```
# Use port 3051 instead of a random port
RemoteAuxPort = 3051
```

2.1.52. TcpRemoteBufferSize

Buffer size for send and receive buffers of both the client and server for TCP/IP connections.

Configurability

Global, client-side, and per-connection

Syntax

```
TcpRemoteBufferSize = integer
```

Unit

Byte

Range

1448 - 32767

Default

8192

The engine reads ahead of the client and can send several rows of data in a single packet. With a larger buffer size, more data can be sent per request (e.g. per fetch).

The server-side configuration only configures the server buffer sizes, and the client-side (and per-connection) configuration only configures the client buffer sizes.

Set TcpRemoteBufferSize to the maximum value for the server.

The buffer size influences — among other things — how many rows a single fetch can return. Current Firebird versions will return at most 10 rows, or however many rows fit into 16 packets (whichever is higher), even if a larger number of rows is requested by the client.

Using WireCompression can also increase the number of rows returned per fetch, if the row data compresses well.

Example

```
# Use maximum buffer size of 32767
TcpRemoteBufferSize = 32767
```

2.1.53. TcpNoNagle

Enables the TCP_NODELAY socket option (which disables "Nagle's algorithm").

Configuration

Global, client-side, and per-connection

Syntax

```
TcpNoNagle = Boolean
```

Default

1 (enabled)

Disabling "Nagle's algorithm" (enabling TCP_NODELAY, the default) can improve the responsiveness of network connections, as the socket will not try to coalesce small packets.

Example

```
# Disable TCP_NODELAY (so, use Nagle's algorithm)
TcpNoNagle = 0
```

2.1.54. TcpLoopbackFastPath

Enables the "TCP Loopback Fast Path" feature (SIO_LOOPBACK_FAST_PATH).

Configuration

Global, client-side, and per-connection

Availability

Deprecated 3.0.11, 4.0.3

Removed 5.0

Syntax

```
TcpLoopbackFastPath = Boolean
```

Default

0 (disabled) — since Firebird 3.0.11 and 4.0.3
1 (enabled) — in older versions

This setting only has effect on Windows 8 and higher and Windows Server 2012 and higher. Microsoft now discourages its use and advises not to use it as it can cause problems with kernel-level filter drivers on socket connections.

In response, Firebird has deprecated this setting in Firebird 3.0.11 and 4.0.3 — changing its default to `0` (disabled), and removed it in Firebird 5.0.

The "TCP Loopback Fast Path" can improve performance of localhost connections on Windows systems. For it to work, both the client and the server need to enable the `SIO_LOOPBACK_FAST_PATH` option on the socket before connecting.

As this option is now deprecated, and Microsoft advises not to use it, exercise caution with this option, and leave or set it disabled.

Example

```
# Disable the "TCP Loopback Fast Path"
TcpLoopbackFastPath = 0
```

2.1.55. IPv6V6Only

When enabled, sets the `IPV6_V6ONLY` socket option on the listener socket.

Configuration

Global

Syntax

```
IPv6V6Only = Boolean
```

Default

`0` (disabled)

By default, the Firebird server listens on the zero IPv6 address (`::`) and accepts all incoming connections, whether IPv4 or IPv6, and `IPv6V6Only` is set to false (`0`). If it is set to true (`1`), the server, still listening implicitly or explicitly on the zero IPv6 address, will accept only IPv6 connections.

A different listening address, either IPv4 or IPv6, can be set using the `RemoteBindAddress` parameter. If an IPv4 address or a non-zero IPv6 address is used, the `IPv6V6Only` setting has no effect.

When enabled, this will also reject IPv4-mapped IPv6 addresses as a value of `RemoteBindAddress`.

Example

```
# Enable IPV6_V6ONLY
IPv6V6Only = 1
```

See also

RemoteBindAddress

2.1.56. RemoteBindAddress

Network interface of the TCP/IP listener.

Configuration

Global

Syntax

```
RemoteBindAddress = string
```

Default

Empty (bind to ::)

The setting accepts an IPv4, IPv6 address, or a hostname; the value may optionally be enclosed in square brackets.

By default — when not specified, or explicitly empty — Firebird listens on :: (the zero IPv6 address). The exact behaviour of the zero IPv6 address depends on IPv6V6Only:

- if disabled (the default), Firebird listens on all IPv4 and IPv6 network interfaces of the host.
- if enabled, Firebird only listens on all IPv6 network interfaces of the host (not IPv4).

It is not possible to specify multiple bind addresses.

If IPv6V6Only is enabled, IPv4-mapped IPv6 addresses are rejected.

When using a hostname, that hostname is used to resolve an IP address, preferring IPv6 over IPv4; the resolved address is then used to bind the server. So, make sure the hostname resolves to one IP address only (i.e. not be multihomed or otherwise multiplexed, nor be both IPv4 and IPv6). Otherwise, the server may bind to a different IP address on each restart (e.g. if the DNS resolution does a round-robin), and clients may have connection failures when they resolve a different IP address than the one the server was bound to.

Examples

```
# listen on IPv4 localhost
RemoteBindAddress = 127.0.0.1

# listen on IPv6 localhost
RemoteBindAddress = ::1

# listen on all IPv4 network interfaces of the host
RemoteBindAddress = 0.0.0.0

# listen on the IP address of fbserver.local
RemoteBindAddress = fbserver.local
```

See also

IPv6V6Only

2.1.57. LockMemSize

Allocation size of the shared memory of the lock manager table.

Configuration

Global, and per-database

Syntax

```
LockMemSize = integer
```

Unit

Byte

Default

64K (64 KiB)

The lock manager table is allocated per database.

The value of LockMemSize is the initial size and the allocation unit. The shared memory will grow dynamically as needed in increments of this allocation unit, up to 2 GiB - 1, or when the system runs out of memory.

Example

```
# Allocate lock memory in increments of 2 MiB
LockMemSize = 2M
```

2.1.58. LockAcquireSpins

The number of times that the lock manager will try acquiring a lock in a loop before waiting.

Configuration

Global, and per-database

Syntax

```
LockAcquireSpins = integer
```

Default

0

The lock table of a database can only be accessed by one server process or thread at a time, and this is governed by a mutex. This mutex can be requested conditionally or unconditionally. A conditional request either succeeds immediately or fails and must be retried. An unconditional

request will make the process or thread wait (block) until the request succeeds.

This setting is only relevant for SuperClassic and Classic.

It is hard to give specific advice for configuring LockAcquireSpins. On multicore machines — when using Classic and SuperClassic — it can be advantageous to spin on the lock a few times before waiting unconditionally, on the other hand if the lock table is heavily contested, this is likely to "burn" CPU cycles. As such, the advice is to experiment with it, and tune it per database in databases.conf.

Example

```
# Spin 5 times while acquiring lock
LockAcquireSpins = 5
```

2.1.59. LockHashSlots

Number of hash slots for locks.

Configuration

Global, and per-database

Syntax

```
LockHashSlots = integer
```

Range

101 - 65521

Default

8191

Use of prime numbers is highly recommended. Setting out of range values will use the boundary values.

The number of hash slots determines how locks are distributed in the lock hash table. If two or more locks are assigned to the same hash slot, they are chained together, and the server must "walk" this chain to find the right lock in the slot.

If the *average* length of lock chains is very long (e.g. 20 locks on average), performance of the server may be affected. In that case, you can increase the number of LockHashSlots to a higher prime number. You can use fb_lock_print to determine the average lock chain length (called "Hash lengths" in the lock print output).

Example

```
# Use 9001 hash slots
LockHashSlots = 9001
```

2.1.60. EventMemSize

Allocation size of the shared memory for the event manager.

Configuration

Global, and per-database

Syntax

```
EventMemSize = integer
```

Unit

Byte

Default

64K (64 KiB)

The memory for the event manager is allocated per database.

The value of EventMemSize is the initial size and the allocation unit. The shared memory will grow dynamically as needed in increments of this allocation unit, until the system runs out of memory.

Example

```
# Allocate event manager memory in increments of 128 KiB
EventMemSize = 128K
```

2.1.61. SnapshotsMemSize

Allocation size of shared memory for snapshot management.

Configuration

Global, and per-database

Availability

Added 4.0

Syntax

```
SnapshotsMemSize = integer
```

Unit

Byte

Default

64K (64 KiB)

The memory for snapshots is allocated per database. Each active snapshot uses 16 bytes of memory.

The value of SnapshotsMemSize is the initial size and the allocation unit. The memory will grow dynamically as needed in increments of this allocation unit, until the system runs out of memory.

Snapshots are used to provide a stable view on data within a transaction or during statement execution. See also Commit Order for Capturing the Database Snapshot in the *Firebird 4.0 Release Notes*.

Example

```
# Allocate snapshot memory in increments of 128 KiB
SnapshotsMemSize = 128K
```

2.1.62. TipCacheBlockSize

Block size of shared memory allocated for TIP cache.

Configuration

Global, and per-database

Availability

Added 4.0

Syntax

```
TipCacheBlockSize = integer
```

Unit

Byte

Default

4M (4 MiB)

The TIP cache is a list of all known transactions with associated Commit Numbers (*CN*), which are used to determine record visibility of active transactions.

The memory for the TIP cache is allocated per database. Each transaction uses 8 bytes of memory. The default size of a TIP cache block is 4 MiB, providing capacity for 512 * 1024 transactions.

A reason to reduce this value is if you have a small TIP cache (or, a small difference between OIT and Next Transaction) and want to conserve memory. A reason to increase this value is if you need a very large TIP cache (or, a very large difference between OIT and Next Transaction) and approach limits on kernel objects allocated for each block (files, mutexes, etc).

The TIP cache is implemented as an array indexed by transaction ID, with as value the corresponding Commit Number. This array is split into blocks of TipCacheBlockSize bytes containing the CN's for all transactions between the OIT and Next Transaction markers. When Next Transaction moves beyond the range of the highest block, a new block is allocated. The oldest block is released when the OIT moves beyond the range of that block.

Example

```
# Block size of 1 MiB (128 * 1024 transactions per block)
TipCacheBlockSize = 1M
```

2.1.63. OutputRedirectionFile

File to redirect stdout and stderr output of server.

Configuration

Global

Syntax

```
OutputRedirectionFile = path
```

Default

`/dev/null` (Linux and other Unix-like OSes)
`nul` (Windows)

By default, stdout and stderr are redirected to the "null-device" to suppress any output.

When set to empty or -, the stderr and stdout are not redirected. When set to a file, it is redirected to that file.

This can be helpful when debugging plugins, UDRs, or UDFs that write to stdout or stderr.

Example

```
# Disable default redirect
OutputRedirectionFile =
# Redirect to file /var/tmp/firebirdoutput.log
OutputRedirectionFile = /var/tmp/firebirdoutput.log
```

2.1.64. CpuAffinityMask

Which CPUs should be used (Windows Only)

Configuration

Global

Syntax

```
CpuAffinityMask = integer
```

Default

`0` (all CPUs)

Sets which processors can be used by the server. The value is taken from a bitmap in which each bit represents a CPU. Thus, to use only the first processor (CPU 0), the value is 1. To use both CPU 0 and CPU 1, the value is 3 (binary 11). To use CPU 1 and CPU 2, the value is 6 (binary 110). The default value is 0 - no affinity will be set.

In Firebird 5 and later, on Windows 10 and later, if affinity is not set by `CpuAffinityMask`, nor by the caller process, then the server tries to exclude efficiency cores from its own affinity mask, i.e. default affinity mask includes performance cores only.

On 32-bit Windows, this can configure at most 32 CPUs; on 64-bit Windows, at most 64 CPUs. On systems with more than 64 processors, it can only configure the processors in the processor group assigned to the Firebird process.

For technical information, see the Microsoft documentation of `SetProcessAffinityMask`.

Example

```
# Use CPU 7 only (binary 10000000)
CpuAffinityMask = 128
# Use CPU 15 and CPU 7 (binary 1000000010000000)
CpuAffinityMask = 32896
```

2.1.65. GCPolicy

Garbage collection policy.

Configuration

Global, and per-database

Syntax

```
GCPolicy = { cooperative | background | combined }
```

Default

`combined` (SuperServer)
`cooperative` (Classic/SuperClassic)

The garbage collection policy configures how Firebird SuperServer performs garbage collection. For Classic and SuperClassic, this setting is ignored; they always use the *cooperative* policy.

cooperative When a connection encounters garbage — old record versions that are no longer interesting — while reading a record, it will collect that garbage immediately. If there is a lot of garbage, this can reduce performance of statement execution and fetching.

background When a connection encounters garbage while reading a record, it will queue the record for the garbage collector. The garbage collector will — at a later time — collect garbage on that record.

combined A combination of *cooperative* and *background*. Some types of garbage will be queued for the garbage collector, while others will be collected by the connection itself.

Example

```
# Use background garbage collection policy
GCPolicy = background
```

2.1.66. MaxStatementCacheSize

Maximum amount of memory per connection for the statement cache.

Configuration

Global, and per-database

Availability

Added 5.0

Syntax

```
MaxStatementCacheSize = integer
```

Unit

Byte

Default

`2M` (2 MiB)

The statement cache is a per-connection cache of previously compiled SQL statements. Setting the value to `0` disables the cache.

The cache is maintained automatically; cached statements are invalidated when required, e.g. when a DDL statement is executed, or when the cache size exceeds `MaxStatementCacheSize`.

Example

```
# Disable statement cache
MaxStatementCacheSize = 0
# Cache 5 MiB
MaxStatementCacheSize = 5M
```

2.1.67. SecurityDatabase

Location of the security database.

Configuration

Global, and per-database

Syntax

```
SecurityDatabase = { path | alias }
```

Default

`$(dir_secDb)/securityN.fdb` (with *N* the major version of Firebird)

The security database is used for authentication and certain global configuration and user privileges (e.g. global authentication mapping, and `CREATE DATABASE` privileges). When using embedded mode, no authentication is performed.

The `SecurityDatabase` can be configured per database in `databases.conf`. Such a security database can be created as a normal database. Initialization as a security database happens per authentication plugin, the first time that authentication plugin is used to create a user. This initialization must be done with an embedded connection otherwise you cannot connect, or authenticated with a `SYSDBA` or `RDB$ADMIN` user of an already initialized authentication plugin.

It is possible to configure a database as its own security database, but only *after* the database has been created.

Example in `databases.conf`

```
# Configure employee example database as its own security database
employee = $(dir_sampleDb)/employee.fdb
{
  SecurityDatabase = employee
}
```

2.1.68. MaxParallelWorkers

Maximum number of parallel workers per database per process.

Configuration

Global

Availability

Added 5.0

Syntax

```
MaxParallelWorkers = integer
```

Range

1 - 64

Default

1 (no parallelism)

Values higher than 1 configure the total number of parallel workers that can be created within a single Firebird process for each attached database. Workers are accounted for each attached database independently.

Parallel workers are used to parallelize certain database operations. In Firebird 5.0, that includes index creation and rebuilding, and sweep.

In SuperServer, worker attachments are implemented as light-weight system attachments, while in Classic and SuperClassic they look like normal user attachments. All worker attachments are embedded into the creating server process, so in Classic there are no additional server processes. Worker attachments are present in monitoring tables. Idle worker attachments are destroyed after 60 seconds of inactivity. In Classic, worker attachments are destroyed immediately when the user connection detaches from the database.

Setting out of range values will use the boundary values.

This number is **per database** and **per process**.

This means that if you use SuperServer or SuperClassic, and set `MaxParallelWorkers` to 8, and you serve 10 databases, then in theory you can have 80 parallel workers (8 per database). Similarly, if you use Classic, and you have 10 connections to a single database, then you could have 80 parallel workers in total (8 per process, each connection is a process).

Example

```
MaxParallelWorkers = 8
```

See also

`ParallelWorkers`

2.1.69. ParallelWorkers

Default number of parallel workers for a connection.

Configuration

Global

Availability

Added 5.0

Syntax

```
ParallelWorkers = integer
```

Range

1 - `MaxParallelWorkers`

Default

1 (no parallelism)

This configures the default number of parallel workers that can be created by a single connection. This default can be overridden with the `isc_dpb_parallel_workers` connection property (up to the maximum `MaxParallelWorkers`).

For accounting purposes, the user connection itself is counted as a worker, which is why value `1` means no parallelism, and a value of `2` means *1 additional worker connection* next to the user connection.

Parallel workers are allocated on a first come, first served basis, so — in SuperServer and SuperClassic — when multiple connections to a database ask for parallel workers, at most `MaxParallelWorkers` workers can be created. For example, if `ParallelWorkers` is `3` and `MaxParallelWorkers` is `4`, and three connections ask for parallel workers, only `4` are allocated, e.g. two workers for the first, two for the second, and none for the third connection.

A connection only asks for parallel workers when performing a parallel operation. Once the operation is complete, the worker is released.

Parallel workers are used to parallelize certain database operations. In Firebird 5.0, that includes index creation and rebuilding, and sweep.

Setting out of range values will use the boundary values.

Example

```
ParallelWorkers = 4
```

See also

`MaxParallelWorkers`

2.1.70. GuardianOption

Configures behaviour of the Firebird Guardian service (Windows only).

Configuration

Global

Syntax

```
GuardianOption = integer
```

Default

1 (always restart)

Supported values

`0` only start the engine/service once

1 always restart the engine/service if it terminates

This setting only has effect when the Firebird Guardian service is installed and running.

The Firebird Guardian service is deprecated, and its use is not recommended. The normal service recovery options provided by Windows should be preferred.

Example

```
GuardianOption = 0
```

2.1.71. ProcessPriorityLevel

Priority level/class for the server process (Windows only).

Configuration

Global

Syntax

```
ProcessPriorityLevel = integer
```

Default

`0` (normal priority)

Supported values

`< 0` low priority (sets `IDLE_PRIORITY_CLASS`)

`0` normal priority (sets `NORMAL_PRIORITY_CLASS`)

`> 0` high priority (sets `HIGH_PRIORITY_CLASS`); equivalent to using the `-b[oostpriority]` commandline option of `instsvc.exe`

Configures the process priority on Windows, which is used by the CPU scheduler to decide how and when to schedule threads of the Firebird process(es).

Setting a negative value may reduce responsiveness of the Firebird server if the system is otherwise busy, and setting a positive value may result in Firebird starving other process of CPU cycles. In general, do not set this to anything other than `0`, and if you do set it, carefully test if your system responds correctly.

For technical information, see the Microsoft documentation of `SetPriorityClass`.

Example

```
# Set high priority
ProcessPriorityLevel = 1
# Set idle priority
```

```
ProcessPriorityLevel = -1
```

2.1.72. IpcName

Name of the shared memory area used as the transport channel for the XNET (a.k.a. local) protocol (Windows only).

Configuration

Global, client-side, and per-connection

Syntax

```
IpcName = string
```

Default

FIREBIRD

Server-side, the configuration is used to create the shared memory to accept XNET connections, and for creating XNET connections to other servers. Client-side, the configuration is used to open the shared memory to make XNET connections to the server.

The user running the server process should have the SE_CREATE_GLOBAL_NAME privilege (or SeCreateGlobalPrivilege), so the shared memory area can be registered in the Global\ namespace. If the user does not have this privilege, it will be registered in the local namespace, and only processes running in the same session will be able to connect using XNET.

If you're running Firebird as a service, and the user does not have SeCreateGlobalPrivilege, processes on the same machine will not be able to connect with XNET, as the service runs in its own session.

You can configure this value if you have multiple Firebird servers running on the same machine which all need to be able to accept XNET connections. In that case, clients will need to be configured with that IpcName as well, either in their firebird.conf, or through isc_dpb_config.

Example

```
IpcName = FIREBIRD_2_5
```

2.1.73. RemotePipeName

Name of the pipe used as a transport channel in the WNET (a.k.a. NetBEUI) protocol (Windows only).

Configuration

Global, client-side, and per-connection

Availability

Removed 5.0

Syntax

```
RemotePipeName = string
```

Default

`interbas`

Server-side, the configuration is used to create the pipe to accept WNET connections, and for creating WNET connections to other servers. Client-side, the configuration is used to open the pipe to make WNET connections to the server.

You can configure this value if you have multiple Firebird servers running on the same machine which all need to be able to accept WNET connections. In that case, clients will need to be configured with that `RemotePipeName` as well, either in their `firebird.conf`, or through `isc_dpb_config`.

Example

```
RemotePipeName = FIREBIRD_2_5
```

2.1.74. UseLegacyKernelObjectsNames

Configures how Firebird creates named Windows kernel objects, such as events, memory mapped files, etc. (Windows only).

Configuration

Global

Availability

Added	4.0.3
Deprecated	4.0.3
Removed	5.0

Syntax

```
UseLegacyKernelObjectsNames = Boolean
```

Default

`false`

Since version 4.0.3, Firebird creates named kernel objects in a private namespace. This allows processes to interact within different Windows sessions, such as a user session and a service session. Also, it uses the engine version in some shared event names; this allows a single process to host Firebird engines of different versions.

This setting is for backward compatibility between Firebird 4.0 releases only, and is not present in Firebird 5.0 and higher. Enabling it allows simultaneous running of processes of newer (>= 4.0.3)

and older (< 4.0.3) subreleases of Firebird 4.0.

Example

```
# Enable legacy behaviour
UseLegacyKernelObjectsNames = true
```

2.1.75. Redirection

Allow multi-hop connections to other Firebird servers.

Configuration

Global

Availability

Deprecated 5.0

Syntax

```
Redirection = Boolean
```

Default

`0` (false)

A multi-hop connection allows you to redirect a connection to a database through intermediate servers.

Deprecated in Firebird 5, to be removed in Firebird 6

This feature was partially broken in Firebird 3 with early (in remote listener) user authentication, and some plugins cannot be used with `Redirection`. We did not receive any related bug reports, i.e. it's unused. Therefore, `Redirection` is declared deprecated in Firebird 5 and will be removed in Firebird 6.

With this setup, a connection string specifies a list of servers to connect through, where the last server is the server that actually hosts the database.

For example, with the connection string `host1:host2:database`, the client will connect to `host1` with the database `host2:database`. If `host1` has redirection enabled, it will — **without authentication on `host1`** — forward the connection to `host2` to open database `database` — where the connection will be authenticated. Packets will be routed through `host1` between `host2` and the client.

In effect, `host1` serves as a pass-through proxy to `host2`.

Security recommendation

Do not enable this feature unless you really know what you're doing!

Careless use of this feature may allow for circumventing security measures like firewalls and other access control measures. For example if `host1` is public facing,

> while host2 should only be internally accessible, but is accessible from host1, with redirection enabled, host2 can also be accessed from the internet as long as people know the hostname.

Example

```
# Explicitly disable redirection
Redirection = 0
```

2.1.76. ServerMode

Configures database access, process, and caching behaviour of the Firebird engine.

Configuration

Global

Syntax

```
ServerMode =
  { Super | ThreadedDedicated
  | SuperClassic | ThreadedShared
  | Classic | MultiProcess }
```

Default

Super

Supported values

Super/ThreadedDedicated

Databases are opened by a single server process with exclusive access. User attachments are processed by threads launched from a common pool, and all share a single database page cache inside the process.

Generally known as “SuperServer”.

SuperClassic/ThreadedShared

Databases are opened by a single server process with shared access. Databases may be opened by multiple processes (including embedded). User attachments are processed by threads launched from a common pool, each having its own database page cache.

Generally known as “SuperClassic”.

Classic/MultiProcess

For each attachment to the server, a separate process is started. Databases are opened with shared access, and may be opened by multiple processes (including embedded). Each attachment (process) has its own database page cache.

Generally known as “Classic”.

The ServerMode affects how Firebird opens databases, and the memory use per attachment. There is no single “best” choice.

Historically, for Firebird 2.1 and older, SuperServer did not scale well with multiple CPU cores. This was improved in Firebird 2.5, and fully addressed in Firebird 3.0.

In other words, CPU scalability is no longer a reason to not use SuperServer.

Example

```
ServerMode = SuperClassic
```

2.1.77. ExtConnPoolSize

Maximum number of inactive (idle) connections in the external connections pool.

Configuration

Global

Availability

Added 4.0

Syntax

```
ExtConnPoolSize = integer
```

Range

`0 - 1000`

Default

`0` (disabled)

The external connections pool is a connection pool for the external data source (EDS) subsystem. For details, consult Pooling of External Connections in the *Firebird 3.0 Release Notes*.

As the connection pool is per process, for Classic this means it is effectively per connection.

Example

```
ExtConnPoolSize = 20
```

2.1.78. ExtConnPoolLifeTime

The number of seconds a connection can be idle in the external connections pool before it is closed.

Configuration

Global

Availability

Added 4.0

Syntax

```
ExtConnPoolLifeTime = integer
```

Unit

Second

Range

1 - 86400 (1 second to 24 hours)

Default

7200 (2 hours)

When an external connection is returned to the idle list of the pool, a timer is started. At or after this timer reaches ExtConnPoolLifeTime, the connection will be closed and removed from the pool.

The timer is stopped and cleared when the connection is returned to the active list of the pool.

The external connections pool is a connection pool for the external data source (EDS) subsystem. For details, consult Pooling of External Connections in the *Firebird 3.0 Release Notes*.

Example

```
# 1.5 hours
ExtConnPoolLifeTime = 5400
```

[1] Unless it reference the *fbclient* of your Firebird installation

[2] Think of denial of service by filling the wrong disk until it runs out of space, or by creating a database that another program will use *if* it exists, allowing unintended data-exfiltration, or by creating a file that causes a program to change its behaviour, checking if files exist or not, etc.

[3] WNET is Windows-only, and was removed in Firebird 5.0

[4] *gsec* and the user management service API have been deprecated since Firebird 3.0 and may be removed in a future version, use user management statements instead

[5] not supported by Legacy_Auth

[6] This depends on the specific implementation of the database encryption plugin, secure solutions are possible

Chapter 3. databases.conf

The file databases.conf defines aliases for databases, and — optionally — database specific configuration. It can be used both server-side, and client-side.

You can define multiple aliases for a single database, but only one of those aliases can be a scoped value (that is, have configuration between braces).

3.1. Server-side usage

When used server-side, the file defines aliases that the server redirects to a specific database file, and — if specified — database-specific configuration used by the server for the database file.

The configuration specified in the scoped value applies for all connections to that database, not just connections using that specific alias.

Only *per-database* configuration items are supported, other configurations items are ignored.

3.2. Client-side usage

When used client-side, the file defines aliases that the client can use as alternative connection strings, and the configuration the client uses for the connection if the connection string matches that alias, or the value of that alias, *exactly*.

The value of the alias can only be a filename, or a legacy connection string. At this time, modern URL connection strings are not supported. See also firebird#8302.

Only *per-database* **and** *client-side* configuration items are supported, other configurations items are ignored.

It is possible some connection properties are only *per-database* server-side, and global client-side, or vice versa. If you find such a case, please report it on the firebird-documentation repository so we can update the documentation.

3.3. Configuration items

The details of the configuration items allowed in databases.conf are documented in chapter firebird.conf. The following configurations items are supported:

- RemoteAccess
- ExternalFileAccess
- TempTableDirectory
- DefaultDbCachePages
- DatabaseGrowthIncrement
- UseFileSystemCache

- TempCacheLimit
- MaxIdentifierByteLength
- MaxIdentifierCharLength
- InlineSortThreshold
- OptimizeForFirstRows
- OuterJoinConversion
- SubQueryConversion
- AuthServer
- AuthClient
- UserManager
- DefaultProfilerPlugin
- WireCryptPlugin
- KeyHolderPlugin
- Providers
- DeadlockTimeout
- StatementTimeout
- ConnectionIdleTimeout
- OnDisconnectTriggerTimeout
- MaxUnflushedWrites
- MaxUnflushedWriteTime
- ClearGTTAtRetaining
- DataTypeCompatibility
- WireCrypt
- DummyPacketInterval
- RemoteAuxPort
- LockMemSize
- LockAcquireSpins
- LockHashSlots
- EventMemSize
- SnapshotsMemSize
- TipCacheBlockSize
- GCPolicy
- MaxStatementCacheSize
- SecurityDatabase

Chapter 4. plugins.conf

The file plugins.conf contains the configuration for various plugins to Firebird.

The file has two types of configuration items with scoped values:

Plugin Registration of a plugin

Config Configuration of a plugin

4.1. Plugin

The Plugin item contains the registration information of a plugin.

Syntax

```
Plugin = plugin-name
{
  [ Module = module-name ]
  [ RegisterName = register-name ]
  [ Config = config-name ]
  [ ConfigFile = config-filename ]
}
```

Table 2. Plugin configuration parameters

Parameter	Description
plugin-name	Name of the plugin used by Firebird to look up — or, load — the plugin.
module-name	Name of the dynamic library containing the plugin. If not specified, *plugin-name* is used.
register-name	Name of the plugin in the dynamic library. If not specified, *plugin-name* is used.
config-name	Name of the Config item in this plugins.conf containing the plugin-specific configuration.
config-filename	Name of the configuration file with the plugin-specific configuration. The file should use simple key-value syntax as used by firebird.conf.

If both Config and ConfigFile are specified, Config will be used.

If neither Config nor ConfigFile are specified, the configuration will be resolved as follows:

1. If a file module-name.conf exists, that will be used as ConfigFile
2. Otherwise, Config is set to *plugin-name*.

These rules also apply to implicitly loaded plugins.

Registration in plugins.conf is optional

For simple cases, the presence of the plugin library in the `plugins` folder is sufficient for a plugin to work. Those plugins are defined *implicitly*.

Registration in `plugins.conf` is only necessary when:

- The plugin has a different name than the module/library name.

  This can be used if you want to use a different name than the original name, or the module contains multiple plugins with different names.

  For example, in the default `plugins.conf` this is used for the `ChaCha64` plugin, which is part of the `ChaCha` module.

- The plugin requires additional configuration.

  For example, in the default `plugins.conf` this is used to configure the `path` configuration item of the `UDR` plugin.

- You have multiple plugins with the same name; you need to give them different names so Firebird can distinguish between them.

4.2. Config

The `Config` item contains the configuration of a specific plugin.

Syntax

```
Config = config-name
{
  { key-name = value <new-line> }...
}
```

Table 3. Config configuration parameters

Parameter	Description
config-name	Configuration name, as referenced from a `Plugin` item.
key-name	Configuration key; plugin-specific
value	Value for configuration key *key-name*; plugin-specific

Each key-value pair must be on a line of its own.

Supported values of *key-name* and *value* are plugin-specific.

Chapter 5. replication.conf

The author has little experience with Firebird replication, so it's possible that the descriptions here are lacking in detail or practicality.

Feedback, additions, or corrections are highly appreciated. Do not hesitate to contact us on firebird-devel or through the firebird-documentation issue tracker, or submit a pull request to the firebird-documentation repository.

The `replication.conf` file configures replication between databases.

This section serves as a reference for the configuration items in `replication.conf`; it does not cover — nor is it intended to cover — all aspects of Firebird replication. At time of writing, only the *Firebird 4.0 Release Notes* and `doc/README.replication.md`[1] in a Firebird installation document replication.

5.1. Syntax

The syntax of `replication.conf` has two types of configuration:

1. server level (applies to all databases)

   ```
   database
   {
     [<key> = <value>]...
   }
   ```

   This is essentially a scoped value *without* an equals ('=') and simple value

2. database level (applies to a specific database)

   ```
   database = database-path
   {
     [ <key> = <value> <new-line> ]...
   }
   ```

   This is a scoped value.

   The *database-path* must be a valid absolute or relative path; aliases or wildcards cannot be used.

The database-level settings override those set on server-level.

5.2. Configuration items

The following sections list the configuration items supported in the scoped values of

replication.conf.

The configuration items are listed in order of appearance in the default replication.conf file.

Primary side

The following configuration items only affect the primary side (a.k.a. master) of replication.

5.2.1. plugin

Plugin used to perform replication.

Replication type

Synchronous, asynchronous

Availability

Added 4.0

Syntax

```
plugin = plugin
```

Default

empty (use built-in replication)

Leaving/setting this empty will use the built-in (default) replication.

Example

```
# Use plugin CustomReplication for all databases
database
{
  plugin = CustomReplication
}

# Revert to default replication for a specific database
database /path/to/example.fdb
{
  plugin =
}
```

The plugin configuration item only affects the primary side of replication. The replica side cannot be overridden by specifying an alternative plugin.

5.2.2. include_filter

Pattern for tables to include in replication.

Replication type

Synchronous, asynchronous

Availability

Added 4.0

Syntax

```
include_filter = [ <sql-regular-expression> ]

<sql-regular-expression> ::=
  !! See Syntax: SQL Regular Expressions !!
```

Default

empty (all tables of *publication set*)

By default, all tables in the *publication set* of the database are replicated. The `include_filter` can be used to reduce this to a subset (only those matching the `include_filter` — and not matching the `exclude_filter`).

The *publication set* of the database can be configured with `ALTER DATABASE`, `CREATE TABLE` and `ALTER TABLE`. This is not covered by this reference. For further details, see Setting Up Replication in the *Firebird 4.0 Release Notes*.

The `include_filter` uses the SQL regular expression syntax, described in Syntax: SQL Regular Expressions in the *Firebird 5.0 Language Reference*.

Example

```
database /path/to/example.fdb
{
  # Only replicate tables with prefix DAT_, and tables CUSTOMER and ORDER
  include_filter = DAT\_%|CUSTOMER|ORDER
}
```

See also

`exclude_filter`

5.2.3. exclude_filter

Pattern for tables to exclude in replication.

Replication type

Synchronous, asynchronous

Availability

Added 4.0

Syntax

```
exclude_filter = [ <sql-regular-expression> ]

<sql-regular-expression> ::=
  !! See Syntax: SQL Regular Expressions !!
```

Default

empty (exclude nothing)

By default, all tables in the *publication set* of the database are replicated. The `exclude_filter` can be used to reduce this to a subset (exclude those matching the `exclude_filter` — possibly already limited to those matching the `include_filter`).

The *publication set* of the database can be configured with `ALTER DATABASE`, `CREATE TABLE` and `ALTER TABLE`. This is not covered by this reference. For further details, see Setting Up Replication in the *Firebird 4.0 Release Notes*.

The `exclude_filter` uses the SQL regular expression syntax, described in Syntax: SQL Regular Expressions in the *Firebird 5.0 Language Reference*.

Example

```
database /path/to/example.fdb
{
  # Do not replicate tables with prefix DAT_, and tables CUSTOMER and ORDER
  exclude_filter = DAT\_%|CUSTOMER|ORDER
}
```

See also

`include_filter`

5.2.4. log_errors

Configures if all errors and warnings are logged to `replication.log`.

Replication type

Synchronous, asynchronous

Availability

Added 4.0

Syntax

```
log_errors = Boolean
```

Default

`true`

This setting only affects error logging on the primary side. For error logging on the replica side, see verbose_logging.

See also

verbose_logging

5.2.5. report_errors

Configures if replication errors are reported to the client application.

Replication type

Synchronous, asynchronous

Availability

Added 4.0

Syntax

```
report_errors = Boolean
```

Default

false

5.2.6. disable_on_error

Configures if a replication error disables replication.

Replication type

Synchronous, asynchronous

Availability

Added 4.0

Syntax

```
disable_on_error = Boolean
```

Default

true

Primary configuration items of built-in replication

The following configuration items are specific to configure the primary side of the built-in (default) replication. If you use a third-party replication plugin, consult the documentation of your replication plugin for its configuration items.

5.2.7. buffer_size

Size of in-memory buffer to accumulate changes that can be deferred until transaction commit/rollback.

Replication type

Synchronous, asynchronous

Availability

Added 4.0

Syntax

```
buffer_size = integer
```

Unit

Byte

Default

`1048576` (1 MiB)

The bigger this value, the less concurrent disk access (related to journal IOPS) happens.

For synchronous replication, it also affects number of network round-trips between primary and replica hosts. However, a larger buffer results in a longer replication "checkpoint" (delay to synchronize the original database with its replica at commit).

5.2.8. journal_directory

Directory to store replication journal files.

Replication type

Asynchronous

Availability

Added 4.0

Syntax

```
journal_directory = path
```

The journal directory is a *working directory* where the Firebird server stores its journal files for the primary for asynchronous replication.

This setting should not be specified when only using synchronous replication.

The journal directory is for use by the primary **only**; this directory and its contents should not be referenced, accessed or modified by the replica (or anything else).

When asynchronous replication is used, journal files (or, replication segments) are moved to the journal archive directory or using a custom archiving command when they are ready to be replicated to (consumed by) the replica.

See also

journal_archive_directory, journal_archive_command, journal_source_directory

5.2.9. journal_file_prefix

Prefix for replication journal file names.

Replication type

Asynchronous

Availability

Added 4.0

Syntax

```
journal_file_prefix = string
```

Default

empty (use database filename)

The journal filename is generated by concatenating the prefix with a sequential number.

If not specified, the database filename (without path) is used as a prefix.

5.2.10. journal_segment_size

Maximum allowed size for a single replication segment.

Replication type

Asynchronous

Availability

Added 4.0

Syntax

```
journal_segment_size = integer
```

Unit

Byte

Default

16777216 (16 MiB)

See also

journal_archive_timeout

5.2.11. journal_segment_count

Maximum allowed number of full replication segments pending archiving.

Replication type

Asynchronous

Availability

Added 4.0

Syntax

```
journal_segment_count = integer
```

Unit

Replication segments

Default

8

Once this limit is reached, the replication process is temporarily delayed to allow the archiving to catch up. If any of the full segments are not archived within one minute, replication fails with an error.

Zero (0) means an unlimited number of segments pending archiving.

5.2.12. journal_group_flush_delay

Delay, in milliseconds, to wait before the changes are synchronously flushed to the journal (usually at commit time).

Replication type

Asynchronous

Availability

Added 4.0

Syntax

```
journal_group_flush_delay = integer
```

Unit

Millisecond

Default

`0` (no delay)

This setting allows multiple concurrently committing transactions to amortise I/O costs by sharing a single flush operation.

Zero (`0`) means no delay, i.e. “group flushing” is disabled.

5.2.13. journal_archive_directory

Directory to store archived replication segments.

Replication type

Asynchronous

Availability

Added 4.0

Syntax

```
journal_archive_directory = path
```

The journal archive directory is used to store replication segments (journal files) that are ready to be consumed by a replica using asynchronous replication.

Firebird will automatically move replication segments to this directory, unless `journal_archive_command` is defined.

If you have a single replica, the journal archive directory can be used directly by the replica (in `journal_source_directory`). This can — for example — be done by mounting or otherwise sharing the directory between both hosts.

If you have multiple replicas, each replica must have its own copy of the journal archive. Replicas will delete journal files they no longer need, which can conflict with the needs of another replica. Make sure that each journal file created by Firebird in this directory is copied to a location unique to a replica, or use the `journal_archive_command` instead.

The value of this configuration item also defines the `$(archpathname)` macro used in `journal_archive_command`.

This configuration item can be omitted when `journal_archive_command` is specified *without* the `$(archpathname)` macro.

See also

`journal_archive_command`, `journal_source_directory`

5.2.14. journal_archive_command

Program (complete commandline with arguments) that is executed when a replication segment is full and needs archiving.

Replication type

Asynchronous

Availability

Added 4.0

Syntax

```
journal_archive_command = string
```

This program **must** return exit-code zero **only** if archiving has been performed successfully. In particular, it **must** return a non-zero exit-code if the target archive already exists.

Special predefined macros are available:

`$(filename)`
: filename (without path) of the journal segment being archived

`$(pathname)`
: full path of the journal segment being archived; same as `journal_directory` + `$(filename)`

`$(archivepathname)`
: suggested full path for the archived segment; same as `journal_archive_directory` + `$(filename)`

If `journal_archive_command` is used *without* the `$(archivepathname)` macro, the configuration item `journal_archive_directory` can be omitted.

A simple example using standard OS commands for archiving

```
# Linux
journal_archive_command = "test ! -f $(archivepathname) && cp $(pathname)
$(archivepathname)"

# Windows
journal_archive_command = "copy $(pathname) $(archivepathname)"
```

See also

`journal_directory`, `journal_archive_directory`

5.2.15. journal_archive_timeout

Timeout, in seconds, to wait until an incomplete (not full) segment is scheduled for archiving.

Replication type

Asynchronous

Availability

Added 4.0

Syntax

```
journal_archive_timeout = integer
```

Unit

Second

Default

`60` (1 minute)

This configuration item can be used to minimize the replication gap if the database is modified rarely.

Zero (`0`) means no intermediate archiving, i.e. segments are archived only after reaching their maximum size (defined by `journal_segment_size`).

See also

`journal_segment_size`

5.2.16. sync_replica

Connection information to the replica database (for synchronous replication).

Replication type

Synchronous

Availability

Added 4.0

Syntax

```
sync_replica = <connection-specification>

<connection-specification> ::=
    [username[:password]]@<database-url>
  | <database-url>
    '{'
      [ <key> = <value> <new-line> ]...
    '}'

<database-url> ::=
  !! Normal Firebird database URL formats !!
```

Multiple entries are allowed (for different synchronous replicas).

The value of `sync_replica` is either:

1. a simple value with the database URL, optionally prefixed with a username and an optional password, or

2. a scoped value where the simple value is **only** a database URL, and configuration of username and password is done through key-value items within the scope.

To specify a username or password containing `:` or `@`, you must use a scoped value.

Table 4. `sync_replica` *configuration items*

Item	Value
`username`	Username for connecting to the replica
`password`	Password for connecting to the replica
`username_file`	Path of a file containing the username for connecting to the replica; the first non-empty line is the username
`password_file`	Path of a file containing the password for connecting to the replica; the first non-empty line is the password
`username_env`	Name of an environment variable containing the username for connecting to the replica
`password_env`	Name of an environment variable containing the password for connecting to the replica

At most one `username*` and one `password*` key may be specified.

If `username_file` or `password_file` have a relative path, the path is resolved against the Firebird root directory.

Example

```
# As simple value
sync_replica = john:smith@server2:replica

# As scoped value
sync_replica = server2:replica
{
  username = john
  password = smith
}

# Password sourced from file
sync_replica = server2:replica
{
  username = john
  password_file = /path/to/password.txt
}

# Username and password sourced from environment
sync_replica = server2:replica
{
  username_env = REPLICA_USERNAME
  password_env = REPLICA_PASSWORD
```

```
}
```

Replica side

The following settings only affect the replica side (a.k.a. secondary or slave) of replication.

5.2.17. cascade_replication

Configures if changes applied to the replica will be subject to further replication (if any configured).

Replication type

Synchronous, asynchronous

Availability

Added 4.0

Syntax

```
cascade_replicaton = Boolean
```

Default

`false`

This setting only has an effect if the replica is also configured as a primary.

If disabled (the default), changes applied from incoming replication are not replicated. Only changes directly applied to the (read-write) replica (i.e. changes to the database by means other than replication) will be replicated to the “next” replica(s).

If enabled, the changes applied from incoming replication and changes directly applied to the replica will be replicated to the “next” replica(s).

5.2.18. journal_source_directory

Directory to search for the journal files to be applied to the replica.

Replication type

Asynchronous

Availability

Added 4.0

Syntax

```
journal_source_directory = path
```

The replica searches the journal source directory for files to apply.

If there is only a single replica, the directory specified in this configuration item can be the same directory as specified in journal_archive_directory of the primary. This can — for example — be done by mounting or otherwise sharing the directory between both hosts.

If you have multiple replicas, each replica must have its own copy of the journal archive, and use its own directory. Replicas will delete journal files they no longer need, which can conflict with the needs of another replica.

See also

journal_archive_directory, journal_archive_command

5.2.19. source_guid

Filter to limit replication to a particular source database (based on its GUID).

Replication type

Asynchronous

Availability

Added 4.0

Syntax

```
source_guid = <quoted-guid>

<quoted-guid> ::= '"{' guid '}"'
```

By default, asynchronous replication will (try to) apply all journal files — from any primary — that shows up in the journal_source_directory.

This setting can be used to only apply journal files from a specific primary, using its database GUID (e.g. as shown on the header page of gstat -h).

The double quotes and braces around the GUID are required!

Example

```
source_guid = "{8FA6D945-BE7E-4376-86D7-E4AAF3889FAF}"
```

See also

journal_source_directory

5.2.20. verbose_logging

Configures if replication.log contains a detailed log of operations performed by replication.

Replication type

Asynchronous

Availability

Added 4.0

Syntax

```
verbose_logging = Boolean
```

Default

`false`

If disabled (the default), only errors and warnings are logged.

This setting only affects logging on the replica side. For error logging on the primary side, see `log_errors`.

See also

`log_errors`

5.2.21. apply_idle_timeout

Timeout (in seconds) to wait before scanning for new replication segments.

Replication type

Asynchronous

Availability

Added 4.0

Syntax

```
apply_idle_timeout = integer
```

Unit

Second

Default

`10` (10 seconds)

When all existing journal files have been applied, replication waits for the specified duration before checking if new files have arrived in `journal_source_directory`.

See also

`journal_source_directory`

5.2.22. apply_error_timeout

Timeout (in seconds) to wait before retrying queued replication segments after an error.

Replication type

Asynchronous

Availability

Added 4.0

Syntax

```
apply_error_timeout = integer
```

Unit

Second

Default

`60` (1 minute)

After an error when applying a replication segment, replication is paused for the specified duration before retrying.

The server disconnects from the replica database, sleeps for the specified timeout, then reconnects and tries to reapply the latest segments from the point of failure.

[1] This link is to the master branch of the Firebird repository, so it may document unreleased features or changes

Chapter 6. Trace configuration

The trace facility of Firebird can be used to trace (record) various types of actions (e.g. connect, prepare, execute, commit) for databases and services. See also *Trace and Audit Services* in the *Firebird 2.5 Release Notes*. Beware, the configuration syntax changed between Firebird 2.5 and Firebird 3.0.

Server administrators can use the *system trace* to trace actions for services, all databases, or a subset of databases. The system trace can be configured in a configuration file, whose path is set in configuration item `AuditTraceConfigFile` in `firebird.conf`. A default Firebird installation contains a file `fbtrace.conf` that shows the layout and configuration items for a system trace; this file is **not** used by default (the default value of `AuditTraceConfigFile` is empty, meaning no system trace).

Any user can start and manage a *user trace* using the services API. An ordinary user — not an administrator, and without the `TRACE_ANY_ATTACHMENT` system privilege — can request a trace only on their own connections, and cannot manage trace sessions started by other users. Administrators can manage any user trace session.

The configuration of a user trace is sent as a string to the services manager, with the same syntax as the system trace configuration file.

The actual trace output is handled by a plugin, configured in `TracePlugin` in `firebird.conf`. This documentation assumes the default trace plugin included with Firebird, `fbtrace`.

6.1. Trace configuration syntax

The trace configuration has two main keywords.

`services`

The `services` keyword only has a Scope; that is, it does not have `= simple-value`. This default section contains the entire trace configuration for all services.

`database`

The `database` keyword is a scoped value. It is used to configure tracing databases.

The `database` keyword can occur without `= simple-value` and only a scope — the default section. The default section contains the trace configuration items for all databases, and establishes defaults that can be overridden by a later occurrence of a `database` section with a simple value.

The simple value of a non-default database section is either a database filename without path or a SQL regular expression pattern which is matched against the absolute path of the database file.

Firebird 2.5 used a different syntax for trace configuration. This reference does not cover that syntax.

The order of evaluation and description of configuration items (but **not** their syntax) are generally the same — assuming the configuration item existed in that version.

6.1.1. Order of evaluation

A trace configuration has the following order of evaluation rules to match a database or service

- the configuration is processed from top to bottom
- configuration items of a default section are applied for all databases or services
  - only one default section of each type is allowed
  - for a service, search ends
  - for a database, search continues
- if a database name matches the pattern of a non-default section
  - its configuration items are applied
  - search ends

In other words, there can only be one (default) `services` section, and one default `database` section. At most one other `database` section (with value) is used for a database, the first one to match.

6.2. Trace configuration items

The following are the configuration items available for trace configuration. Some configuration items apply to both trace types (`database` and `services`), others are specific to a type. The applicable types are listed under *Trace type* in each configuration item section.

The configuration items are listed in order of first appearance in the example `fbtrace.conf`.

6.2.1. `enabled`

Enables tracing.

Trace type

`database`, `services`

Syntax

```
enabled = Boolean
```

Default

`false`

6.2.2. `log_filename`

Path of trace log file (system trace only).

Trace type

`database`, `services`

Syntax

```
log_filename = path
```

path is the absolute or relative path of the log file. This configuration item is only available for system trace configuration.

A backslash must be escaped as \\, or — on Windows — you can also use / as the path separator.

For `database` sections, regular expression references (*sed*-like) for substitutions based on the matched pattern (simple value of the section) are supported. `\0` — whole matched string, `\1` … `\9` — regular expression groups (parenthesized subexpressions).

See also

`max_log_size`

6.2.3. `max_log_size`

Maximum size of log files (in megabytes).

Trace type

`database`, `services`

Syntax

```
max_log_size = integer
```

Unit

Megabyte (MiB)

Default

`0` (unlimited)

When the maximum log file size is reached, the log is rotated by renaming it to include the current date and time. A new log file with the name configured by `log_filename` is created.

A value of `0` (zero, the default) means that the file size is unlimited, and no rotation will occur.

This configuration item is only available for system trace configuration. For limiting user trace sizes, specify `MaxUserTraceLogSize` in `firebird.conf`.

See also

`log_filename`, `MaxUserTraceLogSize`

6.2.4. `include_filter`

Pattern for inclusion of trace events in a trace.

Trace type

`database`, `services`

Syntax

```
include_filter = [ <sql-regular-expression> ]

<sql-regular-expression> ::=
  !! See Syntax: SQL Regular Expressions !!
```

What is matched depends on the trace type:

`database` Statement text of a query-related event.

`services` Service name of a service-operation-related event.

Supported service names

- Backup Database
- Restore Database
- Repair Database
- Add User
- Delete User
- Modify User
- Display User
- Database Properties
- Database Stats
- Get Log File
- Incremental Backup Database
- Incremental Restore Database
- Fixup Database after FS Copy
- Start Trace Session
- Stop Trace Session
- Suspend Trace Session
- Resume Trace Session
- List Trace Sessions
- Set Domain Admins Mapping to RDB$ADMIN
- Drop Domain Admins Mapping to RDB$ADMIN
- Display User with Admin Info
- Validate Database

Other event types are not subject to inclusion or exclusion.

See also
`exclude_filter`

6.2.5. exclude_filter

Pattern for exclusion of trace events in a trace.

Trace type
`database`, `services`

Syntax

```
include_filter = [ <sql-regular-expression> ]

<sql-regular-expression> ::=
  !! See Syntax: SQL Regular Expressions !!
```

What is matched depends on the trace type:

`database` Statement text of a query-related event.

`services` Service name of a service-operation-related event.

See Supported service names.

Other event types are not subject to inclusion or exclusion.

See also
`include_filter`

6.2.6. log_connections

Enables logging of database attach and detach events.

Trace type
`database`

Syntax

```
log_connections = Boolean
```

Default
`false`

See also
`log_services`

6.2.7. connection_id

Limits the trace to a specific connection.

Trace type

`database`

Syntax

```
connection_id = integer
```

Default

`0` (all connections)

The value `0` (zero, the default) traces all connections. A non-zero value traces the connection with that specific connection id.

The connection id is the value reported by — for example — `CURRENT_CONNECTION`, and in column `MON$ATTACHMENT_ID` of `MON$ATTACHMENTS`.

It's not possible to specify multiple connection ids.

6.2.8. log_transactions

Log start and end of transactions.

Trace type

`database`

Syntax

```
log_transactions = Boolean
```

Default

`false`

6.2.9. log_statement_prepare

Log statement prepare.

Trace type

`database`

Syntax

```
log_statement_prepare = Boolean
```

Default

`false`

See also

`print_plan`, `max_sql_length`

6.2.10. log_statement_free

Log statement free (close cursor, unprepare statement, drop statement handle).

Trace type
`database`

Syntax

```
log_statement_free = Boolean
```

Default
`false`

6.2.11. log_statement_start

Log statement execution start.

Trace type
`database`

Syntax

```
log_statement_start = Boolean
```

Default
`false`

6.2.12. log_statement_finish

Log statement execution finish.

Trace type
`database`

Syntax

```
log_statement_finish = Boolean
```

Default
`false`

For statements with a cursor (e.g. `SELECT`), a statement is only finished when the last record has been fetched or the cursor is closed.

See also
`time_threshold`

6.2.13. log_procedure_compile

Log procedure compilation.

Trace type
`database`

Availability

Added 5.0

Syntax

```
log_procedure_compile = Boolean
```

Default
`false`

6.2.14. log_procedure_start

Log procedure execution start.

Trace type
`database`

Syntax

```
log_procedure_start = Boolean
```

Default
`false`

6.2.15. log_procedure_finish

Log procedure execution finish.

Trace type
`database`

Syntax

```
log_procedure_finish = Boolean
```

Default
`false`

See also
`time_threshold`

6.2.16. log_function_compile

Log function compilation.

Trace type
database

Availability
Added 5.0

Syntax

```
log_function_compile = Boolean
```

Default
false

6.2.17. log_function_start

Log function execution start.

Trace type
database

Syntax

```
log_function_start = Boolean
```

Default
false

6.2.18. log_function_finish

Log function execution finish.

Trace type
database

Syntax

```
log_function_finish = Boolean
```

Default
false

See also
time_threshold

6.2.19. log_trigger_compile

Log trigger compilation.

Trace type
database

Availability
Added 5.0

Syntax

```
log_trigger_compile = Boolean
```

Default
false

6.2.20. log_trigger_start

Log trigger execution start.

Trace type
database

Syntax

```
log_trigger_start = Boolean
```

Default
false

6.2.21. log_trigger_finish

Log trigger execution finish.

Trace type
database

Syntax

```
log_trigger_finish = Boolean
```

Default
false

See also
time_threshold

6.2.22. log_context

Log context variable change (through RDB$SET_CONTEXT).

Trace type
database

Syntax

```
log_context = Boolean
```

Default
false

6.2.23. log_errors

Log errors.

Trace type
database, services

Syntax

```
log_errors = Boolean
```

Default
false

See also
log_warnings, include_gds_codes, exclude_gds_codes

6.2.24. log_warnings

Log warnings.

Trace type
database, services

Syntax

```
log_warnings = Boolean
```

Default
false

See also
log_errors, include_gds_codes, exclude_gds_codes

6.2.25. include_gds_codes

List of GDS codes of errors and warnings to include in trace.

Trace type

database, services

Syntax

```
include_gds_codes = [<error> [ ',' <error> ]... ]

<error> ::= <gdscode> | <gdscode-symbol>

<gdscode> ::= integer
  !! see column GDSCODE in SQLCODE and GDSCODE Error Codes and Descriptions !!
  !! This table only contains Firebird errors with a symbol !!

<gdscode-symbol> ::= string
  !! see column Symbol in SQLCODE and GDSCODE Error Codes and Descriptions !!
```

Default

Empty (include all errors and warnings)

If include_gds_codes is not set or set to empty, all errors and warnings — not excluded by exclude_gds_codes — are logged.

If include_gds_codes is non-empty, only errors or warnings that contain one of the specified GDS codes in their status vector will be logged.

Example

```
# Include deadlock (335544336), req_sync (335544364), and 335544321 (arith_except)
include_gds_codes = deadlock, req_sync, 335544321
```

See also

log_errors, log_warnings, exclude_gds_codes, SQLCODE and GDSCODE Error Codes and Descriptions

6.2.26. exclude_gds_codes

List of GDS codes of errors and warnings to exclude from trace.

Trace type

database, services

Syntax

```
exclude_gds_codes = [<error> [ ',' <error> ]... ]

<error> ::= <gdscode> | <gdscode-symbol>
```

```
<gdscode> ::= integer
  !! see column GDSCODE in SQLCODE and GDSCODE Error Codes and Descriptions !!
  !! This table only contains Firebird errors with a symbol !!

<gdscode-symbol> ::= string
  !! see column Symbol in SQLCODE and GDSCODE Error Codes and Descriptions !!
```

Default

Empty (do not exclude any error or warning)

If `exclude_gds_codes` is not set or set to empty, no errors or warnings are excluded — though `include_gds_codes` can apply its own restrictions.

If `exclude_gds_codes` is non-empty, any error or warning that contains one of the specified GDS codes in their status vector will not be logged.

Example

```
# Exclude deadlock (335544336), req_sync (335544364), and 335544321 (arith_except)
exclude_gds_codes = deadlock, req_sync, 335544321
```

See also

`log_errors`, `log_warnings`, `include_gds_codes`, SQLCODE and GDSCODE Error Codes and Descriptions

6.2.27. `log_initfini`

Log trace session init (start) and finish.

Trace type

`database`, `services`

Syntax

```
log_initfini = Boolean
```

Default

`false`

6.2.28. `log_sweep`

Log sweep activity.

Trace type

`database`

Syntax

```
log_sweep = Boolean
```

Default
`false`

6.2.29. `print_plan`

Print access path (plan) with SQL statement.

Trace type
`database`

Syntax

```
print_plan = Boolean
```

Default
`false`

The type of plan — legacy or explained — can be controlled with `explain_plan`.

See also
`log_statement_prepare`, `explain_plan`

6.2.30. `explain_plan`

Print legacy plan (`false`) or explained plan (`true`).

Trace type
`database`

Syntax

```
explain_plan = Boolean
```

Default
`false` (legacy plan)

This setting only has effect when `print_plan` is enabled.

See also
`print_plan`

6.2.31. `print_perf`

Print detailed performance info when applicable.

Trace type
`database`

Syntax

```
print_perf = Boolean
```

Default
`false`

6.2.32. `log_blr_requests`

Log BLR (Binary Language Representation) compilation and execution.

Trace type
`database`

Syntax

```
log_blr_requests = Boolean
```

Default
`false`

See also
`print_blr`

6.2.33. `print_blr`

Print BLR (Binary Language Representation) requests.

Trace type
`database`

Syntax

```
print_blr = Boolean
```

Default
`false`

See also
`log_blr_requests`, `max_blr_length`

6.2.34. `log_dyn_requests`

Log dyn compilation and execution.

Trace type

`database`

Syntax

```
log_dyn_requests = Boolean
```

Default
`false`

See also
`print_dyn`

6.2.35. `print_dyn`

Print dyn requests.

Trace type
`database`

Syntax

```
print_dyn = Boolean
```

Default
`false`

See also
`log_dyn_requests`, `max_dyn_length`

6.2.36. `time_threshold`

Only logs `XXX_finish` events if their timing exceeds the threshold.

Trace type
`database`

Syntax

```
time_threshold = integer
```

Unit
Millisecond

Default
`100` (100 milliseconds)

See also
`log_statement_finish`, `log_procedure_finish`, `log_function_finish`, `log_trigger_finish`

6.2.37. max_sql_length

Maximum length of SQL strings in a log record.

Trace type
`database`

Syntax

```
max_sql_length = integer
```

Unit
Byte

Default
`300`

Beware when adjusting `max_XXX` parameters! The maximum length of a log record for one event should not exceed 64KiB.

See also
`log_statement_prepare`

6.2.38. max_blr_length

Maximum length of BLR (Binary Language Representation) in a log record.

Trace type
`database`

Syntax

```
max_blr_length = integer
```

Unit
Byte

Default
`500`

Beware when adjusting `max_XXX` parameters! The maximum length of a log record for one event should not exceed 64KiB.

See also
`print_blr`

6.2.39. max_dyn_length

Maximum length of dyn in a log record.

Trace type
`database`

Syntax

```
max_dyn_length = integer
```

Unit
Byte

Default
`500`

Beware when adjusting `max_XXX` parameters! The maximum length of a log record for one event should not exceed 64KiB.

See also
`print_dyn`

6.2.40. `max_arg_length`

Maximum length of an individual string argument in a log record.

Trace type
`database`

Syntax

```
max_arg_length = integer
```

Unit
Byte

Default
`80`

Beware when adjusting `max_XXX` parameters! The maximum length of a log record for one event should not exceed 64KiB.

See also
`log_statement_start`, `max_arg_count`

6.2.41. `max_arg_count`

Maximum number of query arguments in a log record.

Trace type
`database`

Syntax

```
max_arg_count = integer
```

Unit

Argument (parameter)

Default

`30`

Beware when adjusting `max_XXX` parameters! The maximum length of a log record for one event should not exceed 64KiB.

See also

`log_statement_start`, `max_arg_length`

6.2.42. `log_services`

Enables logging of service attach, detach, and start events.

Trace type

`services`

Syntax

```
log_services = Boolean
```

Default

`false`

See also

`log_connections`

6.2.43. `log_service_query`

Enables logging of service query events.

Trace type

`services`

Syntax

```
log_service_query = Boolean
```

Default

`false`

Chapter 7. Environment variables

Firebird supports a number of environment variables to configure Firebird server and/or *fbclient*.

7.1. Standard environment variables

The following are the environment variables Firebird supports in normal builds. This documentation does not cover environment variables for debug builds.

7.1.1. EDITOR

Default text editor for *isql*.

Configuration

Client

On Linux and other non-Windows OSes, `EDITOR` is only used if `VISUAL` is not set.

Default if neither `VISUAL` *nor* `EDITOR` *is set:*

Windows	`Notepad`
Linux and other OSes	`vi`

This is **not** a Firebird-specific environment variable, but a common convention for applications.

See also

`VISUAL`

7.1.2. FB_EXPECTED_DB

Default value of the “expected database” for Services Manager connections for use of a non-default security database.

Configuration

Client

The value of this environment variable is a database alias or path. It is used — client-side — to set the value of SPB item `isc_spb_expected_db` if it was not explicitly set.

When establishing a Services Manager connection, the server looks up the `databases.conf` entry of the “expected database”, and uses its value of `SecurityDatabase` as the security database. If there is no `databases.conf` entry, or the entry has no value for `SecurityDatabase`, the value from `firebird.conf` is used.

7.1.3. FIREBIRD

Overrides the Firebird root directory.

Configuration

Client, Server/Embedded

The Firebird root directory determines where the server or client reads other configuration (e.g. `firebird.conf`) from. It also serves as the default location for `firebird.msg`[1], if not overridden by `FIREBIRD_MSG` or `ISC_MSGS`

In general, this should only be set or changed to configure clients and embedded instances, not servers. An incorrect root directory for the server may result in the server using incorrect configuration, or not being able to load required plugins and other components.

7.1.4. FIREBIRD_LOCK

Overrides the location of the Firebird lock files and other shared memory files.

Configuration

Server/Embedded

The value must be set to an absolute path on the physical filesystem of the host. The user running Firebird (or its embedded engine) needs to have sufficient privileges to create and delete files and subdirectories.

If there are multiple processes and/or different OS users accessing the same database files, they all need to use the same lock directory, and they all need sufficient privileges to this lock directory. On Linux, this is generally achieved by giving the permissions to the `firebird` group and adding the users to this group.

Incorrect use of FIREBIRD_LOCK can result in database corruption or other forms of data loss.

Database corruption can happen when multiple processes and/or multiple OS users open the same database with different lock directories. When using different lock directories, these processes cannot coordinate their access.

When in doubt, do not set this and use the server defaults.

This risk exists primarily when using Classic or SuperClassic (see also `ServerMode`); SuperServer opens databases with exclusive access.

The default lock directory is:

Windows

Subdirectory `firebird` in the common appdata directory (`CSIDL_COMMON_APPDATA`) — for example `C:\ProgramData\firebird` — with a fallback to the Firebird root directory[2].

Android

`/data/local/tmp/firebird`.

macOS

`/tmp/firebird` of **or** — if sandboxed — subdirectory `firebird` of the temporary directory of the current user.

Linux and other OSes

`/tmp/firebird`

7.1.5. FIREBIRD_MSG

Location of Firebird message files.

Configuration

Client, Server/Embedded

This setting is primarily a client-side setting, but may get used by the server if it formats and prints messages itself.

How Firebird resolves a message file

1. File specified by `ISC_MSGS`
2. Locale-specific file with relative path `intl/locale.msg` in the Firebird message directory (see The Firebird message directory)

   The value of *locale* is derived from the first 10 characters of the last segment of environment variable `LC_MESSAGES` (after the last `/`, or otherwise after the last `\`, otherwise the entire value)
3. File `firebird.msg` in the Firebird message directory (see The Firebird message directory)

The Firebird message directory

- The directory specified by environment variable `FIREBIRD_MSG`, or
- the default Firebird message directory (if `FB_MSGDIR` was defined in the build configuration), or
- the Firebird root directory (see `FIREBIRD`).

See also

`FIREBIRD`, `ISC_MSGS`

7.1.6. FIREBIRD_TMP

Default location for temporary files.

Configuration

Server/Embedded

The directory specified in the `FIREBIRD_TMP` environment variables is used as the default location for temporary files.

The values of configuration items `TempDirectories` and `TempTableDirectory` take precedence.

The specified directory must exist. The user of the Firebird process — or the process running embedded — must have sufficient privileges to create, write, and delete files and directories in that directory.

If `FIREBIRD_TMP` is not specified, Firebird will fall back as follows:

Fallback on Android

1. `TMP` (environment variable)
2. Use `/data/local/tmp`

Fallback on macOS

1. `TMP` (environment variable)
2. Use `/tmp` **or** — if sandboxed — the temporary directory of the current user

Fallback on Linux and other non-Windows OSes

1. `TMP` (environment variable)
2. Use `/tmp`

Fallback on Windows

1. Result of `GetTempPath`

   According to the Microsoft documentation, this determines its location on the first path found in:

   a. `TMP` (environment variable)
   b. `TEMP` (environment variable)
   c. `USERPROFILE` (environment variable)
   d. The Windows directory

2. Use `C:\temp\`[3]

See also

`TempDirectories`, `TempTableDirectory`

7.1.7. ICU_TIMEZONE_FILES_DIR

Override location of ICU timezone data files.

Configuration

Server/Embedded, Client

Most Firebird builds will look for the ICU timezone data files in directory `tzdata` in the Firebird root directory (on Android, in the root directory itself). Builds can have overridden this default location with the `FB_TZDATADIR` define in their build configuration.

If for some reason you need to use an alternative location for these files, you can configure this with environment variable `ICU_TIMEZONE_FILES_DIR`.

In general, updating the timezone database should be done by replacing the `*.res` files in this `tzdata` directory. For more information, see Updating the Time Zone Database in the *Firebird 5.0 Language Reference*.

7.1.8. ISC_INET_SERVER_HOME

Sets the current working directory of the server (non-Windows only).

Configuration

Server

Use of ISC_INET_SERVER_HOME is not recommended

The exact use case of this environment variable is unclear. It may affect how some relative paths are resolved. We recommend not to set it.

The Firebird sources (`remote/inet.cpp`) have the following comment on this environment variable:

> If the environment variable ISC_INET_SERVER_HOME is set, change the home directory to the specified directory. Note that this will overrule the normal setting of the current directory to the effective user's home directory. This feature was added primarily for testing via remote loopback - but does seem to be of good general use, so is activated for the release version.
> 1995-February-27 David Schnepper
>
> — David Schnepper, remote/inet.cpp

The Firebird Book (Second Edition) by Helen Borrie says about this environment variable:

> Sets the working directory for Classic server *ONLY*. Make certain this variable is not set if you are running Superserver or Superclassic.
>
> — Helen Borrie, The Firebird Book (Second Edition)

We can't corroborate that this is for Classic only, but it will reset the current working directory on each connection received by the server, which seems odd to say the least.

7.1.9. ISC_MSGS

Path of the Firebird message file.

Configuration

Client, Server/Embedded

This setting is primarily a client-side setting, but may get used by the server if it formats and prints messages itself.

The ISC_MSGS — if set — must point to a Firebird message file, not to a directory.

For details on resolution of Firebird message files, see How Firebird resolves a message file in FIREBIRD_MSG.

See also
FIREBIRD_MSG

7.1.10. ISC_PASSWORD

Default password for a connection.

Configuration
Client

The value of ISC_PASSWORD is used by *fbclient* as the default value for isc_dpb_password and isc_spb_password when no explicit password was set and isc_dpb_trusted_auth/isc_spb_trusted_auth is not included.

See also
ISC_USER

7.1.11. ISC_PATH

Path for resolving databases with a filename **without** a path component.

Configuration
Server/Embedded

Firebird will resolve filenames **without** a path component in the directory specified by ISC_PATH. This is done only when the name did not match an alias.

To determine if a file has a path component, Firebird checks if a file contains a :, / or \.

If ISC_PATH is **not** set, and DatabaseAccess is set to Restrict, filenames **without** a path component are resolved against its list of directories.

See also
DatabaseAccess

7.1.12. ISC_USER

Default user for a connection.

Configuration
Client

The value of ISC_USER is used by *fbclient* as the default value for isc_dpb_user_name and isc_spb_user_name when no explicit username was set and isc_dpb_trusted_auth /isc_spb_trusted_auth is not included.

See also

ISC_PASSWORD

7.1.13. VISUAL

Default text editor for *isql* (non-Windows OSes only).

Configuration

Client

If VISUAL is not set, the value of EDITOR is checked before using the default.

The environment variable VISUAL is ignored on Windows.

Default if neither VISUAL *nor* EDITOR *is set:*

Windows	Notepad
Linux and other OSes	vi

This is **not** a Firebird-specific environment variable, but a common convention for applications.

See also

EDITOR

[1] Build configuration may define another default location for the message file.

[2] Given the common appdata directory should always be returned, this fallback shouldn't occur in practice, as far as we're aware.

[3] Given the directories checked for GetTempPath, this fallback is unlikely to get used in practice.

Appendices

Appendix A: License notice

The Original Documentation is titled *Firebird Configuration Reference*.

The Initial Writer of the Original Documentation is: Mark Rotteveel.

Appendix B: Document History

The exact file history is recorded in our *git* repository; see https://github.com/FirebirdSQL/firebird-documentation

Revision History

0.7	23-Dec-2025	MR	Initial revision for *Firebird Configuration Reference* covering `firebird.conf`, `databases.conf`, `plugins.conf`, trace configuration, `replication.conf` and environment variables