The ICU Files

For character set management including collation sequences, Firebird uses an external code library - International Components for Unicode” (ICU). Firebird 4 and later also uses this code library as the source of its Time Zone Database. This is used in support of TIME/TIMESTAMP WITH TIME ZONE data types and to convert local times to and from GMT taking into account the time zone and any daylight savings time offsets that need to be applied.

The ICU library is deployed as a DLL or Shared Object (.so) and may be provided as part of the Operating System.

- For Linux distros, the ICU shared objects are always deployed by the distro and are kept up-to-date as part of the normal OS update cycle.
- For Microsoft Windows, the ICU DLLs have been included in the Windows OS from Windows 10 Version 1703 (Creators Update) onwards. They are not present in earlier versions of Windows. However, Firebird will always ignore the Windows ICU files and will instead use the ICU files installed with the database and located in the Firebird installation folder.
- For macOS, the ICU shared objects are provided as part of macOS.

The ICU library needs to be up-to-date in order to correctly translate local times to and from GMT. This is because, from time to time, there are legislative changes to time zones and daylight savings times and these need to recorded in the ICU library.

Note that as the ICU libraries are also used for character set collation sequences, an updated ICU library can also include a change to character set collations and thus may require that any indexes that depend upon an updated collation sequence have to be rebuilt – or a gbak backup/restore cycle is used to rebuild the indexes.

The Firebird Time Zone Database

The ICU library files already contain a Time Zone Database. However, this may not be up-to-date depending on the how quickly the updates are released by (e.g.) Microsoft or the Linux Distro.

In order to avoid having to update the time zone database without also needing a full ICU library update, a Firebird local copy of the time zone database files are held in the <firebird root>/tzdata folder, as a set of '*.res' files. These are used in preference to the time zone database in the ICU DLLs. The time zone database can be updated by simply replacing these files.

When new versions of the time zone database files are released, they are made available at:

https://github.com/FirebirdSQL/firebird/tree/master/extern/icu/tzdata

The file “le.zip” can be downloaded from this page and contains the replacement '*.res' files for use on little endian architectures (e.g. Intel and AMD64 architectures). These have to extracted and the current versions in the <firebird root>/tzdata folder replaced with the updated versions from the zip.

Updating the ICU and Time Zone Database under Linux

ICU library update is performed automatically when an OS update is performed and the update contains an updated ICU shared object.

It is expected that the Firebird 4 packages should also include a “Firebird-tzdata” package containing the latest Firebird local copy of the time zone database. The time zone database should

thus also be automatically updated as part of the normal update cycle with no need for manual update.

Note: the tzdata or timezone package that is part of most Linux distros also contains a time zone database. However, this is not used by the ICU library.

Updating the ICU and Time Zone Database under Windows

Each Firebird incremental release includes the most up-to-date version of the ICU library and time zone database when it is released. However, the time zone database may also need to be updated between Firebird releases. This is not performed automatically under Windows and has to be manually initiated as described above.

Hopefully, an installer package will eventually be made available to automate the process.

Server Side Considerations

The time zone database for both Firebird Server and embedded server installations should be kept up-to-date and the source of the time zone database files regularly checked for updates.

Client Side Considerations

For character set collations, only the Firebird server/embedded server needs access to the ICU library. However, by default, Firebird clients also require access to the ICU library (or tzdata files) for time zone information. In Firebird 4 Beta 1 client local access to the ICU library was mandatory. However, in later versions, this is optional and it is possible for a Firebird client to rely on the server for all time zone computations. This mode is enabled on a per database connection basis by executing the SQL statement “SET BIND OF TIME ZONE TO EXTENDED”.

The downside of this mode of operation is that there is a cost of an extra two bytes overhead in the “over the wire protocol” per TIME/TIMESTAMP with TIME ZONE field value returned. However, the upside is that it avoids having to maintain an up-to-date copy of the client side ICU library. This is likely to be particularly attractive to deployments with many Windows clients (e.g. on locked down laptops).

IBX automatically detects when the ICU is missing on the client side and invokes “EXTENDED mode” on the user's behalf. It is thus recommended that with applications that use IBX on Microsoft Windows clients, the application is deployed without the ICU or tzdata files in order to avoid the need for client side time zone database updates. That is the application installation folder contains in addition to the application executable and support files, only the Firebird client DLL, “firebird.conf” and “firebird.msg” files.

In order to ensure consistent behaviour across both Windows and Linux clients, IBX includes an option to always ignore any client local ICU files i.e. to automatically execute a “SET BIND OF TIME ZONE TO EXTENDED” statement after connecting to a Firebird 4 server regardless of whether or not a client local ICU library is detected.

The only time use of client local ICU files may be attractive is when the two byte overhead per TIME/TIMESTAMP with TIME ZONE field is a concern. However, such cases are unlikely and database designers that are concerned about a two byte overhead may find it more profitable to review the use of four byte integer columns and consider when they could be replaced with two byte smallint columns.