Firebird 3 has a following set of plugin types:

- user authentication related: AuthServer (validates user's credentials on server when use logins), AuthClient (prepares credentials to be passed over the wire) and AuthUserManagement (maintains a list of users on a server in a form, known to AuthServer);
- ExternalEngine controls use of various engines, see README.external_routines;
- Trace plugin is known from Firebird 2.5, but a way how it interacts with engine was changed to match new generic rules;
- encrypting plugins are for network (WireCrypt) and disk (DbCrypt), there is also helper plugin KeyHolder, which is used to help maintaining secret key(s) for DbCrypt;
- and probably the most important kind – Provider. Firebird 3 supports providers as a kind of plugins, which has nothing outstanding compared with others. See README.Providers for more information about providers.

External Engines

Adriano dos Santos Fernandes

The UDR (User Defined Routines) engine adds a layer on top of the FirebirdExternal engine interface with the purpose of

- establishing a way to hook external modules into the server and make them available for use
- creating an API so that external modules can register their available routines
- making instances of routines “per attachment”, rather than dependent on the internal implementation details of the engine

External Names

An external name for the UDR engine is defined as

```
'<module name>!<routine name>!<misc info>'
```

The <module name> is used to locate the library, <routine name> is used to locate the routine registered by the given module, and <misc info> is an optional user-defined string that can be passed to the routine to be read by the user.

Module Availability

Modules available to the UDR engine should be in a directory listed by way of the path attribute of the corresponding plugin_config tag. By default, a UDR module should be on <fbroot>/plugins/udr, in accordance with its path attribute in <fbroot>/plugins/udr_engine.conf.

The user library should include FirebirdUdr.h (or FirebirdUdrCpp.h) and link with the udr_engine library. Routines are easily defined and registered, using some macros, but nothing prevents you from doing things manually.

Note

A sample routine library is implemented in `examples/udr`, showing how to write functions, selectable procedures and triggers. It also shows how to interact with the current attachment through the legacy API.

Scope

The state of a UDR routine (i.e., its member variables) is shared among multiple invocations of the same routine until it is unloaded from the metadata cache. However, it should be noted that the instances are isolated “per session”.

Character Set

By default, UDR routines use the character set that was specified by the client.

Note

In future, routines will be able to modify the character set by overriding the **getCharSet** method. The chosen character set will be valid for communication with the old Firebird client library as well as the communications passed through the FirebirdExternal API.

Enabling UDRs in the Database

Enabling an external routine in the database involves a DDL command to “create” it. Of course, it was already created externally and (we hope) well tested.

Syntax Pattern

```
  { CREATE [ OR ALTER ] | RECREATE | ALTER } PROCEDURE <name>
    [ ( <parameter list> ) ]
    [ RETURNS ( <parameter list> ) ]
    EXTERNAL NAME '<external name>' ENGINE <engine>

  { CREATE [ OR ALTER ] | RECREATE | ALTER } FUNCTION <name>
    [ <parameter list> ]
    RETURNS <data type>
    EXTERNAL NAME '<external name>' ENGINE <engine>

  { CREATE [ OR ALTER ] | RECREATE | ALTER } TRIGGER <name>
    ...
    EXTERNAL NAME '<external name>' ENGINE <engine>
```

Examples

```
create procedure gen_rows (
  start_n integer not null,
  end_n integer not null
) returns (
  n integer not null
) external name 'udrcpp_example!gen_rows'
  engine udr;
```

```
create function wait_event (
  event_name varchar(31) character set ascii
) returns integer
  external name 'udrcpp_example!wait_event'
  engine udr;

create trigger persons_replicate
  after insert on persons
  external name 'udrcpp_example!replicate!ds1'
  engine udr;
```

How it Works

The external names are opaque strings to Firebird. They are recognized by specific external engines. External engines are declared in configuration files, possibly in the same file as a plug-in, as in the sample UDR library that is implemented in `$(root)/plugins`.

```
  external_engine = UDR {
        plugin_module = UDR_engine
  }

  plugin_module = UDR_engine {
        filename = $(this)/udr_engine
        plugin_config = UDR_config
  }

plugin_config = UDR_config {
        path = $(this)/udr
  }
```

When Firebird wants to load an external routine (function, procedure or trigger) into its metadata cache, it gets the external engine through the plug-in external engine factory and asks it for the routine. The plug-in used is the one referenced by the attribute **plugin_module** of the external engine.

Note

Depending on the server architecture (Superserver, Classic, etc) and implementation details, Firebird may get external engine instances “per database” or “per connection”. Currently, it always gets instances “per database”.

External Routines

Author:

Adriano dos Santos Fernandes <adrianosf@uol.com.br>

Syntax:

```
{ CREATE [ OR ALTER ] | RECREATE | ALTER } PROCEDURE <name>
    [ ( <parameter list> ) ]
    [ RETURNS ( <parameter list> ) ]
    EXTERNAL NAME '<external name>' ENGINE <engine>
```

```
{ CREATE [ OR ALTER ] | RECREATE | ALTER } FUNCTION <name>
    [ <parameter list> ]
    RETURNS <data type>
    EXTERNAL NAME '<external name>' ENGINE <engine>

{ CREATE [ OR ALTER ] | RECREATE | ALTER } TRIGGER <name>
    ...
    EXTERNAL NAME '<external name>' ENGINE <engine>
```

Examples:

```
create procedure gen_rows (
  start_n integer not null,
  end_n integer not null
) returns (
  n integer not null
) external name 'udrcpp_example!gen_rows'
  engine udr;


create function wait_event (
  event_name varchar(31) character set ascii
) returns integer
  external name 'udrcpp_example!wait_event'
  engine udr;


create trigger persons_replicate
  after insert on persons
  external name 'udrcpp_example!replicate!ds1'
  engine udr;
```

How it works:

External names are opaque strings to Firebird. They are recognized by specific external engines. External engines are declared in config files (possibly in the same file as a plugin, like in the config example present below).

```
    <external_engine UDR>
        plugin_module UDR_engine
    </external_engine>

    <plugin_module UDR_engine>
        filename $(this)/udr_engine
        plugin_config UDR_config
    </plugin_module>

    <plugin_config UDR_config>
        path $(this)/udr
    </plugin_config>
```

When Firebird wants to load an external routine (function, procedure or trigger) into its metadata cache, it gets (if not already done for the database*) the external engine through the plugin external engine factory and asks it for the routine. The plugin used is the one referenced by the attribute plugin_module of the external engine.

* This is in Super-Server. In [Super-]Classic, different attachments to one database create multiple metadata caches and hence multiple external engine instances.

UDR - User Defined Routines engine

The UDR (User Defined Routines) engine adds a layer on top of the FirebirdExternal interface with these objectives:

- Establish a way to place external modules into server and make them available for usage;
- Create an API so that external modules can register their available routines;
- Make routines instances per-attachment, instead of per-database like the FirebirdExternal does in SuperServer mode.

External names of the UDR engine are defined as following:

'<module name>!<routine name>!<misc info>'

The <module name> is used to locate the library, <routine name> is used to locate the routine registered by the given module, and <misc info> is an user defined string passed to the routine and can be read by the user. "!<misc info>" may be ommitted.

Modules available to the UDR engine should be in a directory listed through the path attribute of the correspondent plugin_config tag. By default, UDR modules should be on <fbroot>/plugins/udr,accordingly to its path attribute in <fbroot>/plugins/udr_engine.conf.

The user library should include FirebirdUdr.h (or FirebirdUdrCpp.h) and link with the udr_engine library. Routines are easily defined and registered using some macros, but nothing prevents you from doing things manually. An example routine library is implemented in examples/plugins, showing how to write functions, selectable procedures and triggers. Also it shows how to interact with the current database through the ISC API.

The UDR routines state (i.e., member variables) are shared between multiple invocations of the same routine until it's unloaded from the metadata cache. But note that it isolates the instances per session, different from the raw interface that shares instances by multiple sessions in SuperServer.

By default, UDR routines use the same character set specified by the client. They can modify it overriding the getCharSet method. The chosen character set is valid for communication with the ISC library as well as the communications done through the FirebirdExternal API.