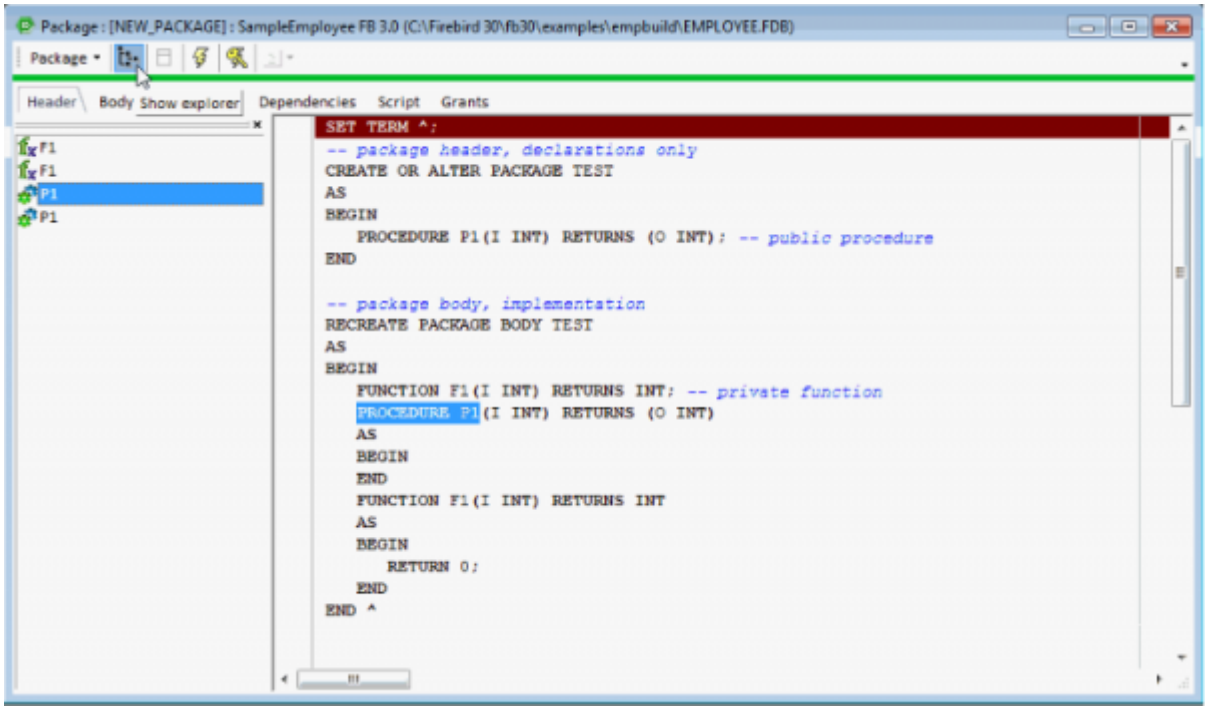
Firebird 3.0 packages

Packages are a new database object type, which allows the logical grouping of stored procedures and functions, similar to Oracle.



In IBExpert (since version 2021.02.09) it is possible to comment and uncomment package procedures and functions. And the [Ctrl] + [C] shortcut in the list of package routines displays a *Copy to clipboard* dialog with a choice of autogenerated statements.

The following is an excerpt from the The Firebird 3.0 Release Notes (29 November 2014 - Document v.0300-16 - for Firebird 3.0 Beta 1) chapter, Procedural SQL (PSQL):

Packages

A. dos Santos Fernandes

Note: This feature was sponsored with donations gathered at the Fifth Brazilian Firebird Developers' Day, 2008.

A package is a group of procedures and functions managed as one entity. The notion of “packaging” the code components of a database operation addresses several objectives:

Modularisation

The idea is to separate blocks of interdependent code into logical modules, as programming languages do.

In programming it is well recognised that grouping code in various ways, in namespaces, units or classes, for example, is a good thing. With standard procedures and functions in the database this is

not possible. Although they can be grouped in different script files, two problems remain:

1. The grouping is not represented in the database metadata.

2. Scripted routines all participate in a flat namespace and are callable by everyone (we are not referring to security permissions here).

To facilitate dependency tracking

We want a mechanism to facilitate dependency tracking between a collection of related internal routines, as well as between this collection and other routines, both packaged and unpackaged.

Firebird packages come in two parts: *a header* (keyword PACKAGE) and a body (keyword PACKAGE BODY). This division is very similar to a Delphi unit, the header corresponding to the interface part and the body corresponding to the implementation part.

The header is created first (CREATE PACKAGE) and the body (CREATE PACKAGE BODY) follows.

Whenever a packaged routine determines that it uses a certain database object, a dependency on that object is registered in Firebird system tables. Thereafter, to drop, or maybe alter that object, you first need to remove what depends on it. As it is a package body that depends on it, that package body can just be dropped, even if some other database object depends on this package. When the body is dropped, the header remains, allowing you to recreate its body once the changes related to the removed object are done.

To facilitate permission management

It is good practice in general to create routines to require privileged use and to use roles or users to enable the privileged use. As Firebird runs routines with the caller privileges, it is necessary also to grant resource usage to each routine when these resources would not be directly accessible to the caller. Usage of each routine to needs to be granted to users and/or roles.

Packaged routines do not have individual privileges. The privileges act on the package. Privileges granted to packages are valid for all package body routines, including private ones, but are stored for the package header.

For example:

```
GRANT SELECT ON TABLE secret TO PACKAGE pk_secret;
GRANT EXECUTE ON PACKAGE pk_secret TO ROLE role_secret;
```

To enable private scope

This objective was to introduce private scope to routines, viz., to make them available only for internal usage within the defining package.

All programming languages have the notion of routine scope, which is not possible without some form of grouping. Firebird packages also work like Delphi units in this regard. If a routine is not declared in the package header (interface) and is implemented in the body (implementation), it becomes a private routine. A private routine can only be called from inside its package.

Signatures

For each routine that is assigned to a package, elements of a digital signature (the set of [routine name, parameters and return type]) are stored in the system tables.

The signature of a procedure or routine can be queried, as follows:

```
SELECT...
-- sample query to come
```

Packaging syntax

```
<package_header> ::=
     { CREATE [OR ALTER] | ALTER | RECREATE } PACKAGE <name>
    AS
    BEGIN
        [ <package_item> ... ]
    END

<package_item> ::=
    <function_decl> ; |
    <procedure_decl> ;

<function_decl> ::=
    FUNCTION <name> [( <parameters> )] RETURNS <type>

<procedure_decl> ::=
    PROCEDURE <name> [( <parameters> ) [RETURNS ( <parameters> )]]

<package_body> ::=
    { CREATE | RECREATE } PACKAGE BODY <name>
    AS
    BEGIN
        [ <package_item> ... ]
        [ <package_body_item> ... ]
    END

<package_body_item> ::=
    <function_impl> |
    <procedure_impl>

<function_impl> ::=
    FUNCTION <name> [( <parameters> )] RETURNS <type>
    AS
    BEGIN
        ...
    END
    |
    FUNCTION <name> [( <parameters> )] RETURNS <type>
```

```
        EXTERNAL NAME '<name>' ENGINE <engine>

<procedure_impl> ::=
    PROCEDURE <name> [( <parameters> ) [RETURNS ( <parameters> )]]
    AS
    BEGIN
        ...
    END
    |
    PROCEDURE <name> [( <parameters> ) [RETURNS ( <parameters> )]]
        EXTERNAL NAME '<name>' ENGINE <engine>

<drop_package_header> ::=
    DROP PACKAGE <name>

<drop_package_body> ::=
    DROP PACKAGE BODY <name>
```

Syntax rules

- All routines declared in the header and at the start of the body should be implemented in the body with the same signature, i.e., you cannot declare the routine in different ways in the header and in the body.
- Default values for procedure parameters cannot be redefined in <package_item> and <package_body_item>. They can be in <package_body_item> only for private procedures that are not declared.

Notes:

- DROP PACKAGE drops the package body before dropping its header.
- The source of package bodies is retained after ALTER/RECREATE PACKAGE. The column RDB$PACKAGES.RDB$VALID_BODY_FLAG indicates the state of the package body. See Tracker item CORE-4487.
- UDF declarations (DECLARE EXTERNAL FUNCTION) are currently not supported inside packages.
- Syntax is available for a description (COMMENT ON) for package procedures and functions and their parameters. See Tracker item CORE-4484.

Simple packaging example

```
SET TERM ^;
-- package header, declarations only
CREATE OR ALTER PACKAGE TEST
AS
BEGIN
   PROCEDURE P1(I INT) RETURNS (O INT); -- public procedure
END

-- package body, implementation
RECREATE PACKAGE BODY TEST
```

```
AS
BEGIN
   FUNCTION F1(I INT) RETURNS INT; -- private function
   PROCEDURE P1(I INT) RETURNS (O INT)
   AS
   BEGIN
   END
   FUNCTION F1(I INT) RETURNS INT
   AS
   BEGIN
      RETURN 0;
   END
END ^
```

[*Note*: More examples can be found in the Firebird installation, in ../examples/package/]

Source: *The Firebird 3.0 Release Notes* by Helen Borrie (Collator/Editor): 29 November 2014 - Dokument v.0300-16 - für Firebird 3.0 Beta 1.

back to top of page

Example using Soundex

Here is an example of a package, based on the Soundex feature we have also demonstrated in the IBExpert documentation chapter, Firebird 3.0 Stored Functions: Example using Soundex.

You can envisage a package as a kind of hidden implementation. For example, here we have a procedure called psoundex and the putils package:

```
ALTER PACKAGE PUTILS
AS
BEGIN
    PROCEDURE PSOUNDEX(
    WORD varchar(1000),
    LNG char(3),
    SLEN bigint = 4)
    RETURNS (SOUNDEX VARCHAR(1000)); -- public procedure
END
```

Now we'll implement the same function name from a package:

```
create or alter function SOUNDEX (
    WORD varchar(1000),
    LNG char(3),
    SLEN bigint = 4)
returns varchar(1000)
AS
declare variable res varchar(1000);
begin
  execute procedure putils.psoundex(:word,:lng,:slen) returning_values res;
```

```
  return res;
end
```

If we add a country-specific version of the Soundex function, we can implement the same functionality and only reference here to the package. The implementation is inside this package.

And we can incorporate another procedure pii, adding to the package header:

```
procedure pii
returns (res numeric(18,16))
```

and then to the package body:

```
procedure pii
returns (res numeric(18,16))
as
begin
  res=3.141592;
  suspend;
end
```

If we then execute the query:

```
select * from putils.pii
```

we get the result:

```
3.141592
```

So you see, you can combine multiple procedures and/or functions into one package, for global implementation.

Firebird 3.0 packages offer you possibilities to make your business logic a little more modular, offer security features, and facilitate permission management and dependency tracking.

back to top of page

New system table RDB$PACKAGES

In Firebird 3.0 a new system table, RDB$PACKAGES, has been added and a new field, RDB$PACKAGE_NAME, added to the existing RDB$PROCEDURES table, to store package metadata.