Blob filter

Blob filters are routines for blobs. They are user-written programs that convert data stored in Blob columns from one subtype to another, i.e. they allow the contents of blob subtype X to be displayed as subtype Y or vice versa. These filters are ideal tools for certain binary operations such as the compression and translation of blobs, depending upon the application requirements.

A blob filter is technically similar to a UDF (user-defined function). It hangs itself in the background onto the database engine, and is used for example to compress the blob, or to specify the format such GIF or JPG (dependent upon use with Windows or Apple Mac). The blob filter mechanism relies on knowing what the various subtypes are, to provide its functionality.

Blob filters are written in the same way that UDFs are written, and are generally part of standard libraries, just as UDFs are.

Declaring a blob filter

A blob filter needs to be explicitly declared in the database before it is used. DECLARE FILTER provides information about an existing Blob filter to the database: where to find it, its name, and the Blob subtypes it works with. First it is necessary to connect to the database using the blob filter, and then issue the statement. The syntax of DECLARE FILTER is as follows:

```
DECLARE FILTER <IB/FB_Filter_Name>
<Parameter_List>
  INPUT TYPE <Type>
  OUPUT TYPE <Type>
  ENTRY_POINT <External_Function_Name>
  MODULE_NAME <Library_Name>;
```

New to Firebird 2.0: Previously, the only allowed syntax for declaring a blob filter was that above. Since Firebird 2.0 there is an alternative new syntax:

```
DECLARE FILTER <name>
  INPUT_TYPE <mnemonic>
  OUTPUT_TYPE <mnemonic>
  ENTRY_POINT <function_in_library>
  MODULE_NAME <library_name>;
```

where <mnemonic> refers to a subtype identifier known to the engine.

Initially they are binary, text and others mostly for internal usage, but it is possible to write a new mnemonic in rdb$types and use it, since it is parsed only at declaration time. The engine keeps the numerical value. Please don't forget that only *negative* subtype values are meant to be defined by users.

To view the predefined types, do

```
select RDB$TYPE, RDB$TYPE_NAME, RDB$SYSTEM_FLAG
  from rdb$types
  where rdb$field_name = 'RDB$FIELD_SUB_TYPE';
```

RDB$TYPE	RDB$TYPE_NAME	RDB$SYSTEM_FLAG
0	BINARY	1
1	TEXT	1
2	BLR	1
3	ACL	1
4	RANGES	1
5	SUMMARY	1
6	FORMAT	1
7	TRANSACTION_DESCRIPTION	1
8	EXTERNAL_FILE_DESCRIPTION	1

Examples can be found at: Declare BLOB subtypes by known descriptive identifiers.

Calling a blob filter

In the same way as UDFs, blob filters can be called from Firebird code whenever an Firebird built-in function call is used. In order to use the blob filter, invoke the FILTER statement when declaring a cursor. Then, whenever Firebird uses the cursor, the blob filter is automatically invoked.

Delete/drop a blob filter

```
DROP FILTER <filter_name>
```

DROP FILTER removes a blob filter declaration from a database. Dropping a blob filter declaration from a database does not remove it from the corresponding blob filter library, but it does make the filter inaccessible from the database. Once the definition is dropped, any applications that depend on the filter will return run-time errors.

A filter can be dropped by its creator, the SYSDBA user, or any user with operating system root privileges.

Text blobs

Firebird 2.0 introduced a number of enhancements for text blobs. Please refer to Enhancements for BLOBs for details.