How to work with Blobs

Counter: 14083
Published: 2007-01-29 16:55:38

Working with BLOB-fields in client InterBase/Firebird applications based on FIBPlus components July, 2006 by Sergey Vostrikov and Serge Buzadzhy

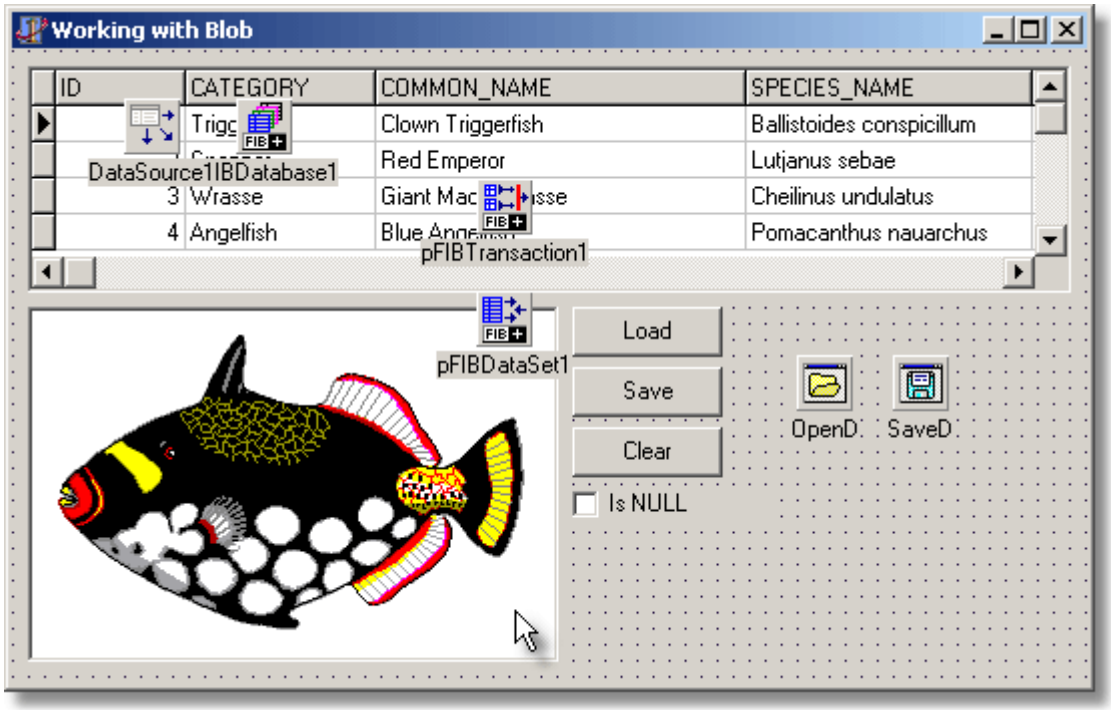
Introduction

There can be advantages in storing non-structured data in your database, such as images, OLE-objects, sounds, etc. For this you will need to use a special data type - BLOB. Before illustrating examples of FIBPlus BLOB-fields, we will consider how server works with BLOBs. It is important to know and remember that in contrast to other fields, BLOBs data are not stored in the table record. Table records store only BLOB_ID, whereas BLOB body is kept in separate database tables. Special IB API functions provide access to the BLOB body. This feature enables developers to store data with undefined size in BLOB fields. Using FIBPlus you do not need to call these functions yourself, as FIBPlus takes care about everything. Anyway, it is useful to know what's going on "behind the curtain".

We will now show you how to use BLOB-fields, using the following table as an example:

```
CREATE TABLE BIOLIFE (
   ID INTEGER NOT NULL,
   CATEGORY VARCHAR (15) character set WIN1251 collate WIN1251,
   COMMON_NAME VARCHAR (30) character set WIN1251 collate WIN1251,
   SPECIES_NAME VARCHAR (40) character set WIN1251 collate WIN1251,
   LENGTH_CM_ DOUBLE PRECISION,
   LENGTH_IN DOUBLE PRECISION,
   NOTES BLOB sub_type 1 segment size 80,
   GRAPHIC BLOB sub_type 0 segment size 80);
```

Using TpFIBDataSet for work with BLOB-fields



Picture. 1. An application form for work with BLOB-fields.
In this example we are using a standard component DBIMage1: TDBImage to show images of the fish stored in (GRAPHIC). Queries for work with BLOB-fields look similar to queries for standard field types:

```
SelectSQL:
   SELECT * FROM BIOLIFE

   UpdateSQL:
   UPDATE BIOLIFE SET
   ID=?NEW_ID,
   CATEGORY=?NEW_CATEGORY,
   COMMON_NAME=?NEW_COMMON_NAME,
   SPECIES_NAME=?NEW_SPECIES_NAME,
   LENGTH__CM_=?NEW_LENGTH__CM_,
   LENGTH_IN=?NEW_LENGTH_IN,
   NOTES=?NEW_NOTES,
   GRAPHIC=?NEW_GRAPHIC
   WHERE ID=?OLD_ID

   InsertSQL:
   INSERT INTO BIOLIFE(
   ID,
   CATEGORY,
   COMMON_NAME,
   SPECIES_NAME,
   LENGTH__CM_,
   LENGTH_IN,
   NOTES,
   GRAPHIC
   )
   VALUES (
   ?NEW_ID,
   ?NEW_CATEGORY,
   ?NEW_COMMON_NAME,
   ?NEW_SPECIES_NAME,
   ?NEW_LENGTH__CM_,
   ?NEW_LENGTH_IN,
   ?NEW_NOTES,
   ?NEW_GRAPHIC
   )

   DeleteSQL:
   DELETE FROM BIOLIFE
   WHERE ID=?OLD_ID

   RefreshSQL:
   SELECT * FROM BIOLIFE
   WHERE
 ID=?OLD_ID
```

Reading nuances:

This is the first "tricky" nuance. «SELECT * FROM BIOLIFE» execution does not read data from BLOB field to the client. It reads only BLOB_ID. Then the following happens "behind the curtain": The DBImage1 component wants to show the field contents of the first record. It refers to pFIBDataSet1 in order to get these contents. Then the component insensibly addresses to the server through IB API functions to get the BLOB body. For this it uses a field Blob_ID from the FIRST record. So you should understand that in the example you fetched to the client only the BLOB field of the first record. On record scrolling in TpFIBDataSet, DBImage1 will refer to data of other records and these references will be sent to the server.

Modification nuances:

BLOB-fields in TFIBDataSet are represented by TBlobField descendants, and thus inherit four special modification methods: LoadFromFile, LoadFromStream, SaveToFile and SaveToStream.

LoadFromFile is used to save data from the external file to the field, LoadFromStream saves any TStream object.

For example if you want to save an image from the external file in a BLOB-field, write the following handler:

```
procedure TMainForm.OpenBClick(Sender: TObject);
begin
 if not OpenD.Execute then
  exit;
 pFIBDataSet1.Edit;
 TBlobField(pFIBDataSet1.FieldByName('GRAPHIC')).LoadFromFile(OpenD.FileName);
 pFIBDataSet1.Post;
end;
```

Pay attention to an important thing: before setting the BLOB-field value you should set pFIBDataSet to the data editing mode. In this case it is pFIBDataSet1.Edit. After loading the data you need to save changes by calling Post.
The second important thing is setting the TBlobField field type. Without this operation FieldByName will return the TField object which lacks necessary methods.

Besides special LoadFromXXX methods, you can also use such simple methods as FieldByName(…).AsString:='asfdsafsadfsad'; to modify BLOB fields.

We can save the value of the BLOB-field to a file or TStream by using SaveToFile and SaveToStream methods:

```
procedure TMainForm.SaveBClick(Sender: TObject);
begin
 if not SaveD.Execute then
  exit;
 if not pFIBDataset1.FieldByName('GRAPHIC').IsNull then
 begin
  TBlobField(pFIBDataSet1.FieldByName('GRAPHIC')).SaveToFile(SaveD.FileName);
 end;
end;
```

Clearing the contents of the field is the same as any other field, i.e:

```
procedure TMainForm.Button1Click(Sender: TObject);
begin
   pFIBDataSet1.Edit;
   pFIBDataSet1.FieldByName('GRAPHIC').Clear;
   pFIBDataSet1.Post;
end;
```

Sometimes you need to know whether the BLOB-field is empty. Using such visual components as TDBImage you cannot get this information for sure. Of course you can make an empty image and save it to BLOB. But you won't know whether there is an image in a BLOB-field. You can also write OnDataChange event handler for the DataSource1: TDataSource component:

```
procedure TMainForm.DataSource1DataChange(Sender: TObject; Field:
   TField);
begin
   CheckBox1.Checked := pFIBDataSet1.FieldByName('GRAPHIC').IsNull;
end;
```

This event is called i.e. when navigating on DBGrid1, so you always know whether the field is empty. And what's going on "behind the curtain"? What's happening when the record with the BLOB-field is being modified?

Variant 1. If the BLOB-field has not been edited, the UPDATE SQL parameter gets the old BLOB_ID. The BLOB-field contents are not sent to the server.

Variant 2. If the BLOB-field has been modified, several operations will be required for writing the new contents. At first IB API functions isc_create_blob2, isc_put_segment, isc_close_blob will save a NEW BLOB body into a database. The client application will know and remember BLOB_ID for this new BLOB. Secondly UPDATE SQL receives the new BLOB_ID, and UPDATE is executed. Thirdly (it's VERY "TRICKY" NUANCE) the server

CHANGES the fetched BLOB_ID in the modified record, so BLOB_ID sent by the client application cannot be used for the second time.

We will make several practical conclusions from the above-mentioned nuances. At first you must use poRefreshAfterPost for TpDataSet where you will modify BLOB-fields (if you have two transactions and no AutoCommit, set the "RefreshTransactionKind" dataset property to "tkUpdateTransaction"). In this case FIBPlus will get BLOB_ID changed by the server and place it instead of the invalid BLOB_ID. Secondly you see that the BLOB-field body is sent to the server BEFORE record modification. If the server will block the recurrent record modification (e.g. by constraints), you will need to send the BLOB body anew for every new modification. This will increase network traffic and database size. That's why we recommend that you separate two processes: modify all non BLOB-fields in one query, and send all BLOB-field changes in a separate query after these modifications are a success.

Note: FIBPlus has a special option for TpFIBDataSet with modifying query auto generation. This option enables developers to separate the two processes: AutoUpdateOptions. SeparateBlobUpdate.

Using TpFIBQuery with BLOBs

If you use TpFIBQuery with BLOB-fields you can use either files or streams (TStream). For example we can write the following procedure, which will save all table images to files:

```
pFIBQuery.SQL: SELECT * FROM BIOLIFE

procedure TMainForm.Button2Click(Sender: TObject);
var
 Index: Integer;
begin
 with pFIBQuery1 do
 begin
  ExecQuery;
  Index := 1;
  while not Eof do
  begin
   FN('GRAPHIC').SaveToFile(IntToStr(Index) + '.bmp');
   Next;
   inc(Index);
  end;
  Close;
 end;
end;
```

Note: The FN method is the short form of FieldByName.
The following code gets all records from the BIOLIFE table, then iterates through them, saves GRAPHIC field value into a file using the SaveFile stream and fetches the next record using the Next method. In the same way we could set the value of the BLOB-parameter:

```
pFIBQuery.SQL: INSERT INTO BIOLIFE (GRAPHIC) VALUES (?GRAPHIC)

procedure TMainForm.Button2Click(Sender: TObject);
var
 Index: Integer;
begin
 with pFIBQuery1 do
 begin
  Prepare;
  for Index := 1 to 3 do
  begin
   Params[0].LoadFromFile(IntToStr(Index) + '.bmp');
   ExecQuery;
  end;
  Transaction.Commit;
```

```
 end;
end;
```

In this example we insert three new records into BIOLIFE and save there images from files "1.bmp", "2.bmp" and "3.bmp".
Note: To make the changes permanent we use the Commit method and you need to restart the application to see record inserted into DBGrid1.

Searching in BLOB-fields

We have considered BLOB-field reading/modification. Now we will illustrate how to search in BLOB-fields. You should understand that if a BLOB parameter is in the where clause, the server compares BLOB_ID of the field and BLOB_ID of the parameter (instead of BLOB-field and BLOB parameter contents). That's why you need to avoid BLOB-parameters and not to use LoadFromFile or LoadFromStream parameters.
As you load parameter values using TStream, actually you create a NEW BLOB with a NEW BLOB_ID at the server. BLOB_ID is TEMPORARY, and is not intended for comparison. That's why the server throws an internal error message when you try to compare a BLOB-field. If you strongly need to compare a BLOB-field with some data, there are two possible variants:

To find records where a BLOB-field is compared with a string of less than 32 Kb:

Set the necessary value to the parameter using AsString. The server will get the SQL_TEXT parameter and then will convert the value necessary for comparison.
For example:

```
select
  ID
from
  BIOLIFE
where
  NOTES = :NOTES
```

The code:

```
begin
with DataSet1 do
 begin
  ParamByName('NOTES').asString:='Sample';
  Open;
 end;
end;
```

To compare a BLOB-field with a value of more than 32 Kb, use a special udf.
For example:

```
select
  ID
from
  BIOLIFE
where
  blobCRC(NOTES) = :NOTES
```

The code :

```
TempStream := TMemoryStream.Create;
Try
 TempStream.LoadFromFile('MyFile');
 with DataSet1 do
 begin
  ParamByName('NOTES') .asInteger:= blobCRCPas(MyStream);
```

```
  Open;
 end;
finally
 FreeAndNil(TempStream);
end;
```

In this example blobCRC is udf, and blobCRCPas is a Pascal function.
Both functions must be identical, that is they must return the same result for the same input data.

The last note (almost obvious): The "magic" number of 32 Kb is a maximum size of CHAR and VARCHAR values.

Unique FIBPlus features: Client BLOB-filters. «Transparent» packing of BLOB-fields.

Many readers already know about BLOB filters technology in Firebird. These user functions enable you to handle (i.e. to pack/unpack, encrypt, etc) BLOB-fields on the server transparently for the client application. This may be useful if you need to pack BLOB-fields in a database without having to change the client program. But this approach will not help you to decrease the net traffic because the server and the application will exchange unpacked fields.

FIBPlus has a mechanism of client BLOB-filters, which is very similar to that in Firebird. The advantage of FIBPlus local BLOB-filter is its ability to considerably decrease application network traffic: BLOB-fields are packed before being sent to the client and unpacked on being sent to the client. You can do this by registering two procedures: for reading and writing BLOB-fields in TpFIBDatabase. FIBPlus will automatically use these procedures to handle all BLOB-fields of the defined type in all TpFIBDataSets using one TpFIBDatabase instance. In this example we will illustrate this mechanism:

First we will create a table with BLOB-fields and a trigger to generate unique primary key values:

```
CREATE TABLE "BlobTable" (

  "Id" INTEGER NOT NULL,

  "BlobText" BLOB sub_type -15 segment size 1);

ALTER TABLE "BlobTable" ADD CONSTRAINT "PK_BlobTable" PRIMARY KEY ("Id");
```

NOTICE THAT sub_type MUST HAVE A NEGATIVE VALUE!

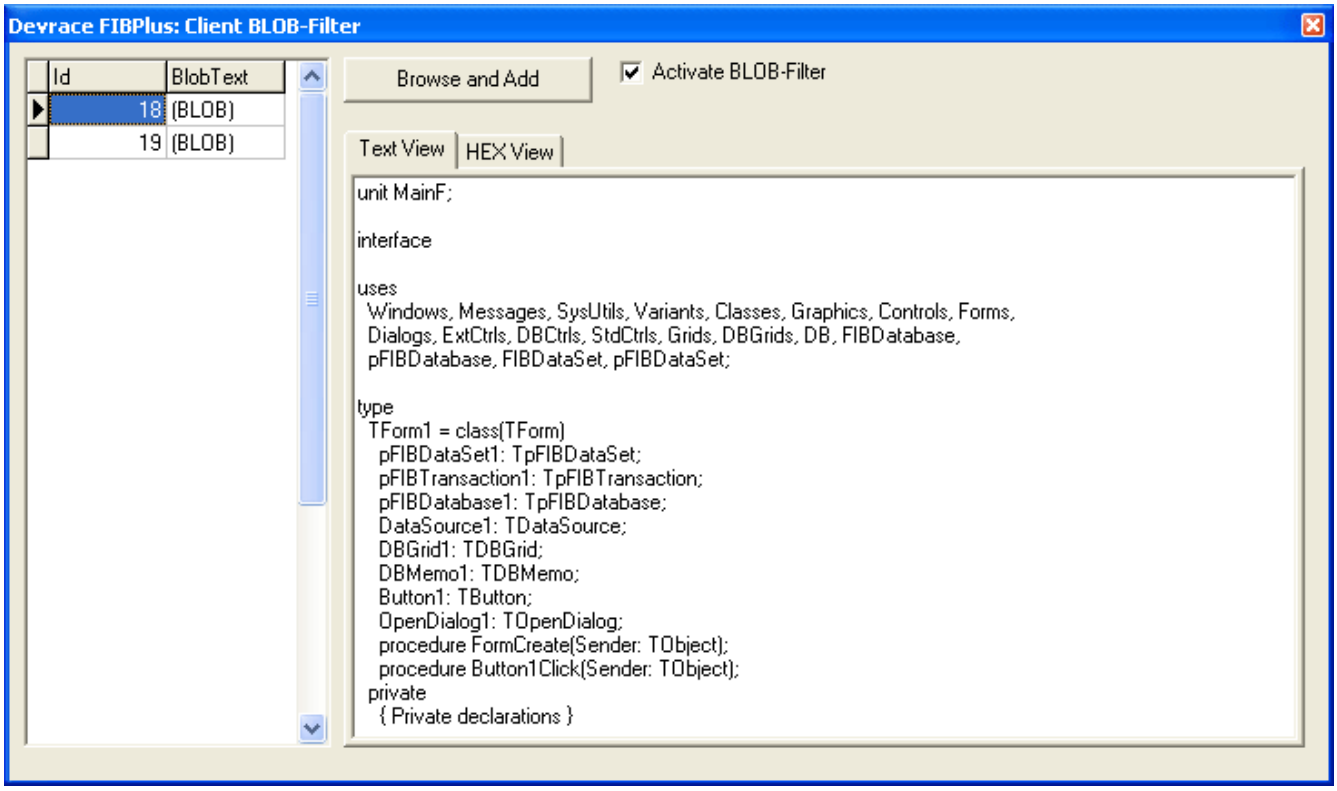
Note: «There are several predefined BLOB subtypes in InterBase. All these subtypes are not negative, e.g. subtype 0 is reserved for binary data, subtype 1 - text, subtype 2 - BLR (Binary Language Representation), etc. Users can also add their own BLOB subtypes with negative values.

Now place the following components on the form:

```
pFIBDataSet1: TpFIBDataSet;

pFIBTransaction1: TpFIBTransaction;

pFIBDatabase1: TpFIBDatabase;

DataSource1: TDataSource;

DBGrid1: TDBGrid;

DBMemo1: TDBMemo;

Button1: TButton;
```

```
OpenDialog1: TOpenDialog;
```

Link FIBPlus components and generate queries for pFIBDataSet1 (only for the "BlobTable" table) with SQL Generator. You will get the following form:



Picture.2. An application with FIBPlus BLOB-filters

We will write a handler for pressing the button:

```
procedure TForm1.Button1Click(Sender: TObject);

begin
 if not OpenDialog1.Execute then
 exit;
  pFIBDataSet1.Append;
TBlobField(pFIBDataSet1.FieldByName('BlobText')).LoadFromFile(OpenDialog1.FileName)
;
  pFIBDataSet1.Post;
end ;
```

Now we will create functions of packing/unpacking blob-fields:

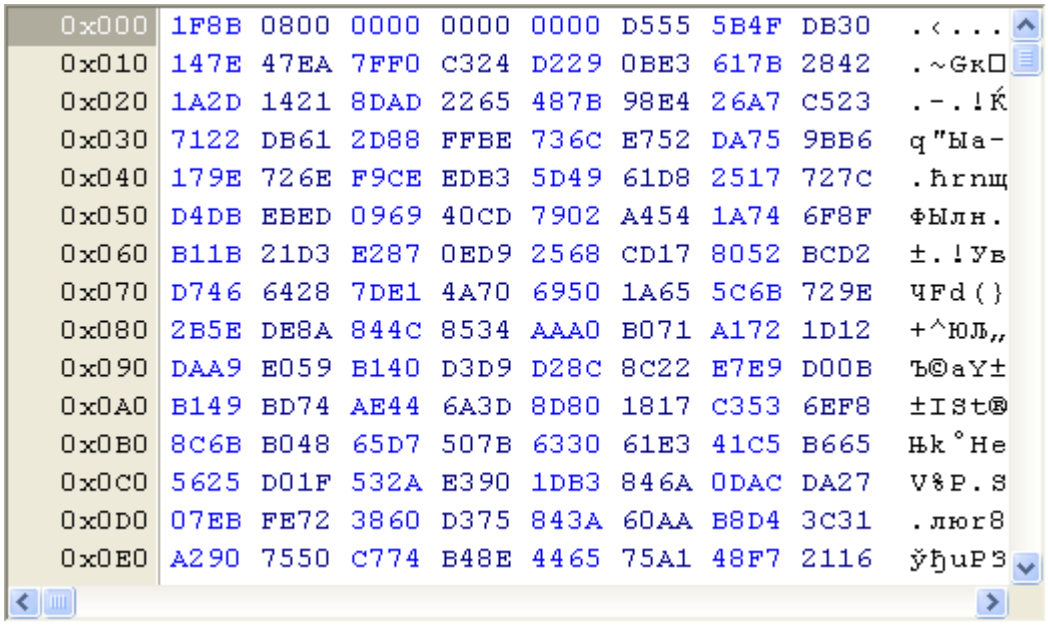
```
procedure  PackBuffer( var  Buffer: PChar; var  BufSize: LongInt);
var  srcStream, dstStream: TStream;
begin
  srcStream := TMemoryStream.Create;
  dstStream := TMemoryStream.Create;
  try
    srcStream.WriteBuffer(Buffer^, BufSize);
    srcStream.Position := 0;
    GZipStream(srcStream, dstStream, 6);
    srcStream.Free;
    srcStream := nil ;
    BufSize := dstStream.Size;
    dstStream.Position := 0;
    ReallocMem(Buffer, BufSize);
    dstStream.ReadBuffer(Buffer^, BufSize);
```

```
  finally
    if  Assigned(srcStream) then  srcStream.Free;
    dstStream.Free;
  end ;
end ;

procedure  UnpackBuffer( var  Buffer: PChar; var  BufSize: LongInt);
var  srcStream,dstStream: TStream;
begin
  srcStream := TMemoryStream.Create;
  dstStream := TMemoryStream.Create;
  try
    srcStream.WriteBuffer(Buffer^, BufSize);
    srcStream.Position := 0;
    GunZipStream(srcStream, dstStream);
    srcStream.Free;
    srcStream:=nil;
    BufSize := dstStream.Size;
    dstStream.Position := 0;
    ReallocMem(Buffer, BufSize);
    dstStream.ReadBuffer(Buffer^, BufSize);
  finally
    if  assigned(srcStream) then  srcStream.Free;
    dstStream.Free;
  end ;
end ;
```

Do not forget to add two modules to the section **uses**: zStream and IBBlobFilter. The first is used to make archives with data, the second controls BLOB-filters and is included in FIBPlus. Now you only have to register BLOB-filters by calling the RegisterBlobFilter function. The value of the first parameter is a BLOB-field type (in this case it is -15); the second and third parameters are functions of BLOB-field packing/unpacking:

```
procedure  TForm1.FormCreate(Sender: TObject);
begin
  pFIBDatabase1.RegisterBlobFilter(-15, @PackBuffer, @UnpackBuffer);
  pFIBDatabase1.Connected := True;
  pFIBDataset1.Active := True;
end ;
```

Run the application, delete some records and add new ones. You will see no changes. But if you look what is really saved in BLOB-fields, you will see that all the data are packed:



Picture 3. Data in the BLOB-field, packed by FIBPlus local filter.

So, if the application sends already packed BLOBs to (and gets from) the server, network traffic can considerably decrease! Of course you can pack BLOB-fields without using the above-described mechanism of BLOB-filters. For example, you can compress a field in the Button1Click procedure before saving it; then decompress in the AfterScroll handler (or do some similar operations). But, firstly, you will greatly simplify your code using the centralized mechanism of BLOB-filters (as BLOB fields are handled imperceptibly for the rest parts of the program) and secondly you will avoid commonplace errors (e.g. when you have packed BLOB fields in one part of the program and no packed BLOBs in another).

Note:

If you write filtered BLOBs in stored procedures, you must set the subtype of the input parameter in the stored procedure. For example:

```
CREATE PROCEDURE "BlobTable_U"(
 "Id" INTEGER,
 "BlobText" BLOB SUB_TYPE -15)
AS
BEGIN
  UPDATE "BlobTable"
  SET "BlobText" = :"BlobText"
  WHERE ("Id" = :"Id");
END;
```

In case you do not set the input parameter subtype, the BLOB-parameter will have default subtype 0 and no filtering will happen in the client application on calling this stored procedure.

Translated by Marina Novikova.

Special thanks to Jason Chapman for proof-reading.