1 Storing and Loading FLOBs

This document describes, how a FLOB can be stored (read) to (from) disk. Basically, the C++ datastructure of a FLOB consists of the root block containing a reference to the data.

If a FLOB is stored, we have to handle two different cases. If the FLOB is a proper LOB, i.e. its data size exceeds a threshold, the data are stored separately from FLOB's root block. Otherwise, the data are stored together with it.

1.1 Storing a FLOB

The function `storeDB` stores the `DBArray db` into the `SmiRecord record` at offset `offset`. If `db` is a proper LOB, the data are stored into a file with id `fileid`. If the file id is not known, use `0` as default value. If several `DBArrays` are to be stored, use `0` only for the first one and the "returned" value for the next ones, e.g.

```
...
SmiFileId fileid = 0;
storeDb(db1, record, offset, fileid);
storeDb(db2, record, offset, fileid);
...
```

Note: The following function works only for `DBArrays` with subtype `int`. For a `DBArray` of other types, just change the corresponding code. For a simple `FLOB` (not a `DBArray`), replace `DBArray<...>` by `FLOB` and `GetFLOBSize` by `GetSize`.

```
void storeDb(
          DBArray<int> db,      // array to store
          SmiRecord& record,    // target for the root
                                // and possible the data
          size_t& offset,       // offset
          SmiFileId& fileid ){

   if(db.IsLob()){
      db.SaveToLob(fileid);
   }

   // store the root of the DBArray
   int size = sizeof(DBArray<int>);
   record.Write(&db, size, offset);
   offset += size;

   // store non-lob-data
   if(!db.IsLob()){
      int extSize = db.GetFLOBSize();
      if(extSize>0){
         // copy data to a byte block in memory
         char* extElem = (char *) malloc(extSize);
         db.WriteTo(extElem);
         // store the data to the record
         record.Write(extElem, extSize, offset);
         offset += extSize;
         free(extElem);
      }
   }
}
```

1.2 Loading a FLOB

Within the representation of a C++ class in addition to members some function pointers are stored to realize virtual functions. After a restart of the application, these pointers may point to addresses different to the addresses used during the preceding execution. But on disk, we have stored the "old" values. To correct them, we cast the root block using a variant of the `new` operator.

If the datasize is small, i.e. the FLOB is not a LOB, we have to load the data directly from the record. Otherwise the data will be loaded automatically on demand.

```
void readDb( DBArray<int>& db,    // DBArray to read
             SmiRecord& record,   // record containing db
             size_t& offset)      // position of db
{

  // read the root from the record
  int size = sizeof(DBArray<int>);
  record.Read(&db,size,offset);
  offset += size;

  // cast to correct some pointers
  new (&db) DBArray<int>;

  // if the array is no lob, read the data directly
  if(!db.IsLob()){
     unsigned int extSize = db.GetFLOBSize();
     char* data = (char*) malloc(extSize);
     record.Read(data, extSize, offset);
     offset += extSize;
     db.ReadFrom(data);
  }
}
```