DlisReader Library

DlisReader is a library for reading information from files written in DLIS (RP66) format version 1. The library exposes extern functions by which user application gets DLIS file records. Full list of exported functions see in appendix A. First group of functions work with DLIS file:

```
/* File handling */
extern "C" __declspec(dllexport) DLIS_RESULT Open(LPCSTR FileName)
extern "C" __declspec(dllexport) void Close()
```

`Open` function opens DLIS file and reads it into a tree-shaped structure as it is shown in DlisBrowser program. After opening the application reads desired records, processes them and calls `Close` function which closes the file and deletes used variables in memory.

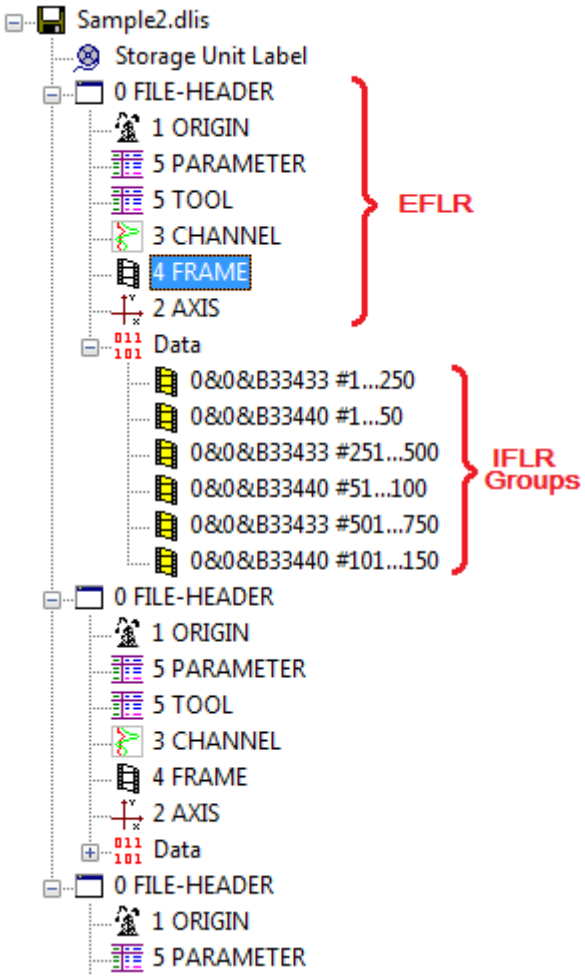

Figure 1

Each node in the tree represents a DLIS logical record described in RP66 specification http://w3.energistics.org/RP66/V1/Toc/main.html chapter 3. The second group of exported functions is used to navigate through the tree:

```
/* Tree handling */
extern "C" __declspec(dllexport) DLISTREEHANDLE Root()      // get root

extern "C" __declspec(dllexport) DLIS_RESULT ChildCount(    // get count of children in the node
       /*in*/DLISTREEHANDLE node,
       /*out*/int* n)

extern "C" __declspec(dllexport) DLIS_RESULT Child(         // get n-th child of the node
       /*in*/DLISTREEHANDLE node, int n,
       /*out*/DLISTREEHANDLE* child)

extern "C" __declspec(dllexport) DLIS_RESULT Parent(        // get parent of the node
       /*in*/DLISTREEHANDLE node,
```

```
        /*out*/DLISTREEHANDLE* parent)

extern "C" __declspec(dllexport) DLIS_RESULT RecordType(   // get record type (Appendix 2, fig. A-2)
        /*in*/DLISTREEHANDLE node,
        /*out*/BYTE* bIFLR, BYTE* type, LPCSTR* set_type)
```

You pass into functions a node handle of type `DLISTREEHANDLE` and get handle of its child, root, etc. The functions return error code, full list of codes see in appendix B.

There are two kinds of records (see 3.1):

- an Explicitly Formatted Logical Record (EFLR)
- an Indirectly Formatted Logical Record (IFLR)

Function `RecordType` returns flag meaning is record IFLR or not, type of the record as it described in RP66 specification Appendix A, fig A-2, and set type in textual form. For example, Frame record (selected in Fig. 1) returns `bIFLR`=0, `type`=4 and `set_type`="FRAME". Some records return special value of type:

SUL (Storage Unit Label) returns -1
Encrypted record returns -2
Data (Group of IFLR records) returns -3

Note that Data Group record is not a part of DLIS format, it is created by DlisReader for convenient navigation through the tree and fast access to IFLR records. Data Group is node containing all IFLRs of current logical file.

EFLR records handling

Each EFLR encapsulates a table of information. The rows of the table represent *Objects*, and the columns of the table represent *Attributes* of the *Objects*. At the beginning of the table there is a *Template* that defines the columns.

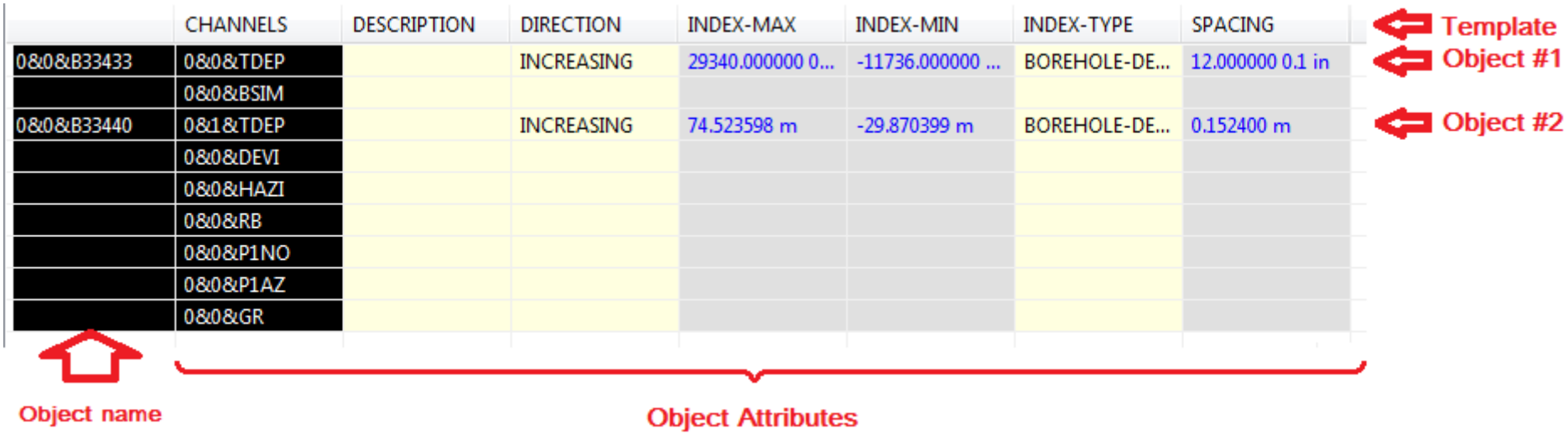

Figure 2

The following group of functions is designed for *Template* and *Object* handling.

```
/* Object handling */

extern "C" __declspec(dllexport) DLIS_RESULT TemplateCount( // get count of template attributes
       /*in*/DLISTREEHANDLE node,
       /*out*/int* n)

extern "C" __declspec(dllexport) DLIS_RESULT TemplateName( // get i-th template name and descriptor
       /*in*/DLISTREEHANDLE node, int i,
       /*out*/LPCSTR* name, BYTE* descriptor)

extern "C" __declspec(dllexport) DLIS_RESULT FindTemplate( // find template by name
       /*in*/DLISTREEHANDLE node, LPCSTR name,
       /*out*/int *i, BYTE* descriptor)

extern "C" __declspec(dllexport) DLIS_RESULT ObjectCount(  // get count of objects in the node
       /*in*/DLISTREEHANDLE node,
       /*out*/int* n)

extern "C" __declspec(dllexport) DLIS_RESULT ObjectName(   // get name of j-th object of the node
       /*in*/DLISTREEHANDLE node, int j,
       /*out*/DLIS_OBNAME* name)

extern "C" __declspec(dllexport) DLIS_RESULT Object(       // get j-th object of the node
       /*in*/DLISTREEHANDLE node, int j,
       /*out*/DLISOBJECTHANDLE* name)
```

Object name is represented in OBNAME format (see Appendix B, B.23). `ObjectName` function reads it into `DLIS_OBNAME` structure:

```
struct DLIS_OBNAME
       {
       unsigned long OriginRef;
       unsigned char CopyNumber;
       DLISstring    Identifier;
       };
```

Object function returns object handle in variable of `DLISOBJECTHANDLE` type. It is used by the following group of functions for Attribute handling:

```
/* Attribute handling */

extern "C" __declspec(dllexport) DLIS_RESULT ObjectAttribCount(   // get count  of object attributes
       DLISOBJECTHANDLE hobj,
       int* attr_cnt)

extern "C" __declspec(dllexport) DLIS_RESULT ObjectAttribute(     // get i-th attribute of the object
       DLISOBJECTHANDLE hobj, int i,
       DLIS_ATTRIBUTE* attr)
```

`ObjectAttribute` function reads Attribute into `DLIS_ATTRIBUTE` structure:

```
struct DLIS_ATTRIBUTE
       {
       const char*    Label;
       unsigned long  Count;
       REPR_CODE      ReprCode;
       const char*    Units;
       DLISvalue*     Value;
       };
```

RP66 specification describes Attribute in Fig. 3-5. *Value* is pointer to an array of Attribute values, *Count* is values quantity. `DLISvalue` is structure which can contain data in different *Representation Codes*:

```
typedef BYTE REPR_CODE;
struct DLISvalue
       {
       // Representation codes values
       enum {
           NOCODE =0,  FSHORT, FSINGL, FSING1, FSING2, ISINGL, VSINGL, FDOUBL,
           FDOUB1 =8,  FDOUB2, CSINGL, CDOUBL, SSHORT, SNORM,  SLONG,  USHORT,
           UNORM  =16, ULONG,  UVARI,  IDENT,  ASCII,  DTIME,  ORIGIN, OBNAME,
           OBJREF =24, ATTREF, STATUS, UNITS
           };
```

```
        REPR_CODE ReprCode;
        BYTE*     pData;
        };
```

RP66 specification describes *Representation Codes* in Appendix B. DlisReader provides a number of functions for reading `DLISvalue` structure.

```
/* DLISvalue handling */
extern "C" __declspec(dllexport) long Int(DLISvalue* dv)          // get integer
extern "C" __declspec(dllexport) unsigned long Uint(DLISvalue* dv) // get unsigned integer
extern "C" __declspec(dllexport) float Float(DLISvalue* dv)       // get float
extern "C" __declspec(dllexport) double Double(DLISvalue* dv)     // get double
extern "C" __declspec(dllexport) DLISdate Date(DLISvalue* dv)     // get date
extern "C" __declspec(dllexport) DLISstring String(DLISvalue* dv)  // get string
extern "C" __declspec(dllexport) DLIS_OBNAME Obname(DLISvalue* dv) // get OBNAME
extern "C" __declspec(dllexport) DLIS_OBJREF Objref(DLISvalue* dv) // get OBJREF
extern "C" __declspec(dllexport) DLIS_ATTREF Attref(DLISvalue* dv) // get ATTREF
```

You have to check the `ReprCode` member of `DLISvalue` structure and call the appropriate function of this list.

Encrypted records handling

DLIS file may contain encrypted records. DlisReader cannot show its content. However, if user knows encryption algorithm, a custom dynamic-link library (dll) can be created and DlisReader will use it for decoding. The dynamic-link library shall meet the following requirements:

- its name shall be 'encr###.dll' where ### - Producer's Company Code from Logical Record Segment Encryption Packet (LRSEP) in decimal form (no leading zeroes);
- it shall be located in the same directory where DlisReader.dll is;
- it shall expose function 'Decrypt' with the following prototype:

```
extern "C" bool __declspec(dllexport) Decrypt(const void* ep, BYTE* record, WORD ln)
```

where `ep` - pointer to Logical Record Segment Encryption Packet (LRSEP);
`record` - pointer to encrypted record;
`ln` - length of encrypted record.

If a record is encrypted, DlisReader calls `Decrypt` function, passes encrypted record and its Logical Record Segment Encryption Packet (see 2.2.2.2). The function decrypts the record and returns `true`, if decryption is impossible function returns `false`. After successful decryption the record is processed as common record, if decryption fails *RecordType* function returns -2 for this node.

IFLR records handling

There are two ways access data in IFLR records. With the following functions you can get total count of IFLRs in given frame and get IFLR by its sequential number. But these functions may work slowly because they scan all records in *Data Group* node.

```
/* Data group handling */

extern "C" __declspec(dllexport) DLIS_RESULT IflrTotalCount(// get total count of IFLRs in given frame
        DLISTREEHANDLE node, DLIS_OBNAME* frm_name,
        int* iflr_cnt);

extern "C" __declspec(dllexport) DLIS_RESULT IflrByNumber( // get IFLR by its sequential number
        DLISTREEHANDLE node, DLIS_OBNAME* frm_name, int sn,
        DLISIFLRHANDLE* hiflr);
```

Another way to access IFLR is scanning *Data Group* node with tree navigation functions and using the following functions:

```
extern "C" __declspec(dllexport) DLIS_RESULT IflrGroupCount(// get count of IFLRs in IFLR group and its
frame
        /*in*/DLISTREEHANDLE node,
        /*out*/DLIS_OBNAME* frame, int* cnt);

extern "C" __declspec(dllexport) DLIS_RESULT Iflr(// get j-th IFLR  of the node and its sequential
number
        /*in*/DLISTREEHANDLE node, int j,
```

```
        /*out*/int* sn, DLISIFLRHANDLE* hiflr);
```

Note that before using these functions you have to check the record type by function `RecordType`. Only records of type `FDATA` can be applied by these functions. IFLR type `EOD` mark end of data and does not contain information, type `NOFORM` contains unformatted data, types 128-255 are reserved for Private IFLRs.

On getting IFLR handle of `DLISIFLRHANDLE` type you can request data. In the most cases user wants to get data in `float` format. If so, use `IflrData` function:

```
extern "C" __declspec(dllexport) DLIS_RESULT IflrData(     // get IFLR data in float format
/*in*/DLISIFLRHANDLE hiflr, int chan_count, int chan_nb[], int dv_cnt,
/*out*/float dv[])
```

The following parameters are passed to the function:

`hiflr` – IFLR handle returned by `IflrByNumber` or `Iflr` functions;
`chan_count` – quantity of channels requested from IFLR; in other words, size of `chan_nb` array;
`chan_nb[]` – array containing 0-based indices of requested channels. Array shall be `chan_count` size;
`dv_cnt` – size of output array;
`dv[]` – array receiving channel data. **Note!** If one or more of requested channels have dimension more than 1, the array shall have appropriate size to fit all data. For example, if you request 3 channels with dimensions 1, 16 and 1, you shall provide array with 18 floats, value of the first channel will be placed into `dv[0]`, values of the second into `dv[1]…dv[16]` and last – into `dv[17]`. If size is insufficient, error DLIS_WRONG_INDEX is returned.

Sometimes user needs to get data in its original format. For this purpose can be used `IflrDataVal` function:

```
extern "C" __declspec(dllexport) DLIS_RESULT IflrDataVal(  // get IFLR data in DLISvalue format
       /*in*/DLISIFLRHANDLE hiflr, int chan_count, int chan_nb[],int dv_cnt,
       /*out*/ DLISvalue dv[])
```

The difference with `IflrData` is in last parameter: you pass an array of `DLISvalue` variables which is filled with IFLR data. To process them you can use functions for reading `DLISvalue` structure. **Note**! You shall process the received array before next `IflrDataVal` call because `pData` member points to an inner array of the library. Their values become invalid after next call of `IflrDataVal`.

Appendix A

List of exported functions.

```
/* File handling */
typedef BOOL (*OPEN_FPTR)(const char* filename);
typedef void (*CLOSE_FPTR)();

/* Tree handling */
typedef DLISTREEHANDLE (*ROOT_FPTR)();              // get root node
typedef DLIS_RESULT (*CHILDCOUNT_FPTR)(             // get count of childs in the node
       /*in*/DLISTREEHANDLE node,
       /*out*/int* cnt);
typedef DLIS_RESULT (*CHILD_FPTR)(                  // get n-th child of the node
       /*in*/DLISTREEHANDLE node, int n,
       /*out*/DLISTREEHANDLE* child);
typedef DLIS_RESULT (*PARENT_FPTR)(                 // get parent of the node
       /*in*/DLISTREEHANDLE node,
       /*out*/DLISTREEHANDLE* parent);
typedef DLIS_RESULT (*RECORDTYPE_FPTR)(             // get record type (Appendix 2, fig. A-2)
       /*in*/DLISTREEHANDLE node,
       /*out*/BYTE* bIFLR, BYTE* type, LPCSTR* set_type);

/* Object handling */
typedef DLIS_RESULT (*TEMPLATECOUNT_FPTR)(          // get count of template attributes
       /*in*/DLISTREEHANDLE node,
       /*out*/int* cnt);
typedef DLIS_RESULT (*TEMPLATENAME_FPTR)(           // get i-th template name and descriptor
       /*in*/DLISTREEHANDLE node, int i,
       /*out*/LPCSTR* name, BYTE* descriptor);
typedef DLIS_RESULT (*FINDTEMPLATE_FPTR)(           // find template by name
    /*in*/DLISTREEHANDLE node, LPCSTR name,
    /*out*/int* i, BYTE* descriptor);
typedef DLIS_RESULT (*OBJECTCOUNT_FPTR)(            // get count of objects in the node
       /*in*/DLISTREEHANDLE node,
       /*out*/int* cnt);
typedef DLIS_RESULT (*OBJECTNAME_FPTR)(             // get name of j-th object of the node
       /*in*/DLISTREEHANDLE node, int j,
       /*out*/DLIS_OBNAME* name);
typedef DLIS_RESULT (*OBJECT_FPTR)(                 // get j-th object of the node
       /*in*/DLISTREEHANDLE node, int j,
       /*out*/DLISOBJECTHANDLE* name);

/* Attribute handling */
typedef DLIS_RESULT (*OBJECTATTRIBCOUNT_FPTR)(      // get count of object attributes
       /*in*/DLISOBJECTHANDLE hobj,
       /*out*/int* attr_cnt);
typedef DLIS_RESULT (*OBJECTATTRIBUTE_FPTR)(        // get i-th attribute of the object
       /*in*/DLISOBJECTHANDLE hobj, int i,
       /*out*/DLIS_ATTRIBUTE* attr);

/* Data group handling */
typedef DLIS_RESULT (*IFLRTOTALCOUNT_FPTR)(         // get total number of IFLRs in given frame
       /*in*/DLISTREEHANDLE node, DLIS_OBNAME* frame,
       /*out*/int* cnt);
typedef DLIS_RESULT (*IFLRBYNUMBER_FPTR)(           // get IFLR by its sequential number
       /*in*/DLISTREEHANDLE node, DLIS_OBNAME* frame, int sn,
       /*out*/DLISIFLRHANDLE* hiflr);

/* IFLR handling */
typedef DLIS_RESULT (*IFLRGROUPCOUNT_FPTR)(         // get number of IFLRs in IFLR group and its frame
       /*in*/DLISTREEHANDLE node,
       /*out*/DLIS_OBNAME* frame, int* cnt);
typedef DLIS_RESULT (*IFLR_FPTR)(    // get j-th IFLR  of the node and its sequential number
       /*in*/DLISTREEHANDLE node, int j,
       /*out*/int* sn, DLISIFLRHANDLE* hiflr);
typedef DLIS_RESULT (*IFLRDATA_FPTR)(               // get IFLR data in float format
       /*in*/DLISIFLRHANDLE hiflr, int chan_cnt, int chan_nb[], int dv_cnt,
       /*out*/float dv[]);
typedef DLIS_RESULT (*IFLRDATAVAL_FPTR)(            // get IFLR data in DLISvalue format
       /*in*/DLISIFLRHANDLE hiflr, int chan_cnt, int chan_nb[], int dv_cnt,
       /*out*/DLISvalue dv[]);

/* DLISvalue handling */
typedef long (*INT_FPTR)(/*in*/DLISvalue* dv);              // get integer
typedef unsigned long (*UINT_FPTR)(/*in*/DLISvalue* dv);    // get unsigned integer
typedef float (*FLOAT_FPTR)(/*in*/DLISvalue* dv);           // get float
typedef double (*DOUBLE_FPTR)(/*in*/DLISvalue* dv);         // get double
```

```
typedef DLISdate (*DATE_FPTR)(/*in*/DLISvalue* dv);         // get date
typedef DLISstring (*STRING_FPTR)(/*in*/DLISvalue* dv);     // get string
typedef DLIS_OBNAME (*OBNAME_FPTR)(/*in*/DLISvalue* dv);    // get OBNAME
typedef DLIS_OBJREF (*OBJREF_FPTR)(/*in*/DLISvalue* dv);    // get OBJREF
typedef DLIS_ATTREF (*ATTREF_FPTR)(/*in*/DLISvalue* dv);    // get ATTREF
```

Appendix B

List of error codes.

DLIS_SUCCESS	Successful completion
DLIS_FILE_ERROR	Error on opening or reading DLIS file; call GetLastError() to get file error code
DLIS_TMP_FILE_ERROR	Error in temporary file; call GetLastError() to get file error code
DLIS_FORMAT_ERROR	Common format error in DLIS file
DLIS_WRONG_HANDLE	Wrong handle
DLIS_WRONG_INDEX	Index is beyond array
DLIS_NOT_FOUND	Searched object not found
DLIS_NOT_EFLR	Node is not EFLR
DLIS_NOT_FHLR	Node is not FILE-HEADER
DLIS_NOT_DATAGROUP	Node is not Data Group
DLIS_NOT_IFLRGROUP	Node is not IFLR Group or its type is not FDATA
DLIS_NO_FRAME	Logical file does not have FRAME record
DLIS_NO_CHANNEL	Logical file does not have CHANNEL record
DLIS_NO_CHANNELS_IN_FRAME	FRAME object does not have CHANNELS attribute
DLIS_CHANNEL_NOT_FOUND	A channel object from FRAME record not found in CHANNEL record
DLIS_REPR_CODE_NOT_FOUND	REPRESENTATION-CODE attribute not found in CHANNEL object
DLIS_NO_IFLR_NB	IFLR with requested sequential number not found
DLIS_VERS_NOT_SUPPORTED	DLIS has version which is not supported
DLIS_NOT_LICENSED	Limited functionality of non-licensed library