1 Abstract

Author: Fabian Noetzel
Version: August 2, 1999

This tutorial explains the writing and using of dynamically linked libraries (DLLs) within XploRe. Sometimes, DLLs are called shared libraries or shared objects, however, we will use the expression DLL in this tutorial. First, we show how to use a DLL in XploRe. Second, we explain how to write a function in C/C++ which can be used as a DLL. Third, we give a short summary of XploRe commands for using DLLs.

2 Using the DLL in XploRe - nchs.xpl

To call the headhanging algorithm DLL in XploRe, we use the quantlet nchs.xpl. A quantlet is a kind of procedure or function which executes different commands. The input and output variables of nchs.xpl are listed in the table below.

<table>
<thead>
<tr>
<th>Input variable</th>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nfor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NNfor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NTRIPfor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NITERfor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THETASTARin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xin0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xin1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input variable</td>
<td>Dimension</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>erg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>err</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

nchs.xpl1 looks like this (the numbers do not belong to the quantlet):

1. \( \text{proc} (\text{erg, err}) = \text{bang8} (\text{Nfor, NNfor, NTRIPfor, NITERfor, THETASTARin, xin0, xini, yin, wt}) \)

2. \( \text{erg} = \text{matrix} (\text{rows(yin)}, \text{cols(yin)}) \)

3. \( h = \text{dlopen} ("\text{bang.so}\") \)

4. \( \text{param} = \text{list} (\text{Nfor, NNfor, NTRIPfor, NITERfor, THETASTARin, xin0, xini, yin, wt, erg}) \)

5. \( \text{byrow} = 0|0|0|0|0|0|0|0|0|0 \)

6. \( \text{type} = -8|-8|-8|-8|-8|-8|-8|-8 \)

7. \( \text{opt} = \text{list} (\text{type, byrow}) \)

8. \( \text{err} = \text{dlcallex} (h, "\text{bang8x}\", \text{param}, \text{opt}) \)

9. \( \text{erg} = \text{param.erg} \)

10. \( \text{dlclose} (h) \)

11. \( \text{endp} \)

1. (11) \( \text{proc} (\text{erg, err}) = \text{bang8} (\cdot) \) defines a procedure named \( \text{bang8} \). In brackets you give the input (\( \cdot \)) and output parameters (\( \text{erg, err} \)). Every quantlet starts with

\[
\text{proc} (\text{output variable(s)}) = \text{MyQuantlet(input variable(s))}
\]

and ends with \( \text{endp} \). For a detailed explanation, see FlowControl

2. Then we define our output matrix \( \text{erg} \), which will store the result of the head-banging algorithm. It has the same dimension as \( \text{yin} \)

3. The next step is to open the DLL for use in XploRe

\( h = \text{dlopen} ("\text{bang.so}\") \)

Note that XploRe needs to be able to find this library. From XploRe you can set the path for DLLs by

\[
\text{setenv ("xp4dll", "/your\_absolute\_path\_to\_the\_dll")}
\]

Alternatively you can give the absolute path within the \text{dlopen} command.

4. Every DLL function in XploRe can deal with up to 16 parameters. However, this number should be sufficient, since the arguments can be arbitrarily large arrays.
With the function `list` we build a list of the variables and assign it to `param`.

(5) The data in a matrix can be stored by row or by column. For each parameter `byrow` can take the value zero or non-zero. By default the value zero is used which means the data in a matrix are stored column wise.

Consider the matrix `x` in XploRe which prints as

```
Contents of x
[i,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
```

The pointer `x` in your C/C++ file contains the values

```
x = {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

If the entry in `opt.byrow` is non-zero then the pointer `x` contains the values

```
x = {1, 4, 7, 2, 5, 8, 3, 6, 9}
```

which means the data are stored row wise.

(6) In XploRe, every parameter is numeric or text. However, C/C++ uses much more different data types. With `type`, we tell the DLL which type we want. The following types are possible:

<table>
<thead>
<tr>
<th>XploRe object</th>
<th>C/C++ data type</th>
<th>entry in opt.type</th>
</tr>
</thead>
<tbody>
<tr>
<td>numeric</td>
<td>8 byte float</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>4 byte float</td>
<td>4</td>
</tr>
<tr>
<td>numeric</td>
<td>8 byte integer</td>
<td>-8</td>
</tr>
<tr>
<td></td>
<td>4 byte integer</td>
<td>-4</td>
</tr>
<tr>
<td></td>
<td>2 byte integer</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>1 byte integer</td>
<td>-1</td>
</tr>
<tr>
<td>text</td>
<td>1 byte char</td>
<td>-1</td>
</tr>
</tbody>
</table>

(7) The list `opt` specifies the elements of `param` more and will be passed to our DLL function.

(8) The command `dlcallex` has the following syntax:

```
err = dlcallex (h, "func", param, opt)
```

with `h` a DLL handle, `func` a DLL function (here: `bang8x`, `param` a list of an arbitrary number of parameters and `opt` an optional list. The parameter `err` contains an error code when the execution of the command has failed.

(9) Assign the result to the output variable of `mchs.xpl`

(10) close the DLL
3 The C/C++ Source Code

3.1 The Program - bang8x.c

Here is the source code of bang8x.c. Parts which are not relevant for this tutorial are omitted.

... 

#define ERROR_PARAM 10100 // define error codes which will 
#define ERROR_TYPE1 10201 // be assigned to 'err'
#define ERROR_TYPE2 10202
#define ERROR_TYPE3 10203
...

#include 'dll.h' // ensures that your C/C++ program is 
    // compatible with all the compilers 
    // mentioned in dll.h

// headbanging algorithm code follows 
...

// the DLL function starts

EXTERN int EXPORT bang8x(int argc, int *type, void *ptr)
{
    INT8 *Nforp, Nfor, NNfor, *NNforp, NTRIPfor, *NTRIPforp, NITERfor;
    INT8 *NITERforp;
    int row, iter;

    // assign the elements of 'param' from 'nchs.xpl' to the 
    // adequate variables

    Nforp    = (INT8 *) ptr[0];
    Nfor     = *Nforp;
    NNforp   = (INT8 *) ptr[1];
    NNfor    = *NNforp;
    NTRIPforp = (INT8 *) ptr[2];
    NTRIPfor = *NTRIPforp;
    NITERforp = (INT8 *) ptr[3];
    NITERfor = *NITERforp;
    THETASTARinp = (DBL8 *) ptr[4];
    THETASTARin  = *THETASTARinp;
    xinOp     = (DBL8 *) ptr[5];
    xin1p     = (DBL8 *) ptr[6];
    yinp      = (DBL8 *) ptr[7];
    wtp       = (DBL8 *) ptr[8];
    erg       = (DBL8 *) ptr[9];

    // check if the number of arguments given by the user is correct

    if ( argc != 10)
return(ERROR_PARAM);

// check if the type of the arguments is correct

if (type[0] != -8)
    return(ERROR_TYPE1);
if (type[1] != -8)
    return(ERROR_TYPE2);
if (type[2] != -8)
    return(ERROR_TYPE3);
if (type[3] != -8)
    return(ERROR_TYPE4);
...

// original headbanging code, adapted

if ( Nfor > N || NNfor > NN || NTRIPfor > NTRIP || NITERfor > NITER )
{
    return(CONTR_PARAM_TOO_BIG);
}

for (row = 0; row < Nfor; row++)
{
    xin[row][0] = (float) xinp[row];
    xin[row][1] = (float) xinp[row];
    yin[row] = (float) yinp[row];
    wt[row] = (float) wtp[row];
    yhat[row] = (float) yinp[row];
    temp[row] = (float) yinp[row];
}

find_dists(Nfor, NNfor, NTRIPfor, NITERfor);
first_sort(Nfor, NNfor, NTRIPfor, NITERfor);
find_body_triples(Nfor, NNfor, NTRIPfor, NITERfor, THETASTARin);
find_edge_triples(Nfor, NNfor, NTRIPfor, NITERfor, THETASTARin);
for (iter = 0; iter < NITERfor; iter++)
    { 
        fprintf(stderr, "iter = %d\n", iter);
        fflush(stderr);
        extrapolate(Nfor, NNfor, NTRIPfor, NITERfor, yhat);
        smooth_body_points(Nfor, NNfor, NTRIPfor, NITERfor, yhat);
        smooth_edge_points(Nfor, NNfor, NTRIPfor, NITERfor, yhat);
        update(Nfor, NNfor, NTRIPfor, NITERfor, yhat);
    }

// assign the result to the 'output' variable 'erg'

for (row = 0; row < Nfor; row++)
    erg[row] = (double) yhat[row];
}

// original headbanging code follows

...
The first parameter \texttt{argc} tells us how many parameters are given by the user. If the number of parameters is not 10, \texttt{bangx} returns the value 10100. In our XploRe macro, we examine this error-message with \texttt{error(condition, "string")}.

The second parameter \texttt{type} tells us which type the parameter given to our function \texttt{bangx has}. We check if the type of the parameter given is correct. This is identically to the list given in \texttt{opt.type}.

Note, that we can not take any responsibility for a successful conversion! For example the Symantec compiler does not provide an 8 byte integer. Also the conversion for a float value to a 1 byte integer will fail if the float values are outside the range \(-126\) till \(+127\).

The third parameter \texttt{ptr} is a set of pointers on pointers which contain the data from XploRe, it is the equivalent to \texttt{param}.

Note, that we use the macro \texttt{DBL8} rather than \texttt{double}. \texttt{DBL8} stands for an 8 byte float value which may not be the \texttt{double} data type in all C/C++ compilers. According to the table before we define also \texttt{DBL4}, \texttt{INT8}, \texttt{INT4}, \texttt{INT2} and \texttt{INT1}.

### 3.2 General

A DLL written in C or C++ can contain any sort of functions written in these languages. Note that the parameters of a C or C++ function are called by reference and that the function return value is an integer. Thus, your function head has to be of the following type

\begin{verbatim}
EXTERN int EXPORT MyFunction(.)
\end{verbatim}

Note:

- It is not possible to have more than 16 arguments. However, this number should be sufficient, since the arguments can be arbitrarily large arrays.

- Since all arguments are references, you also have to transfer the dimensions of higher-dimensional XploRe objects.

- The "output" parameters are also arguments of the C/C++ function. It is important to initialize this variable in XploRe before calling the DLL function.

### 3.3 dll.h

The file \texttt{dll.h} is a header file which ensures that your C program is compatible with all the compilers mentioned below. You may put \texttt{dll.h} into your directory where you have your C/C++ sources.

To write a DLL, a deep knowledge of \texttt{dll.h} is not necessary, so you might skip this section and continue with the next one if your compiler is supported by \texttt{dll.h}.

If you are using other compilers you are welcomed to add them in \texttt{dll.h}.
Let us shortly examine the include file dll.h:

```c
#ifdef __cplusplus
    #define EXTERN extern "C"
#else
    #define EXTERN
#endif

///*/*/*/* Compiler depended definitions *//*
///*/*/*/* Definition of EXPORT *//*
///*/*/*/* Definition of nan, inf and math constants *//*
///*/*/*/* Definition of fixed byte datatypes *//*

/**** GNU C++ *//
#ifdef __GNUC__
    #define EXPORT
double dnan = 0.0/0.0, dinf = 1.0/0.0;
#endif

/**** Visual C++ *//
#ifdef _MSC_VER
    #define EXPORT __declspec(dllexport)
    #include <ymath.h>
double dnan = __NaN__D, dinf = __Inf__D;
    #define M_PI 3.14159265358979323846
#endif

#define INT1 signed char
#define INT2 short
#define INT4 int
#define INT8 long
#define DBL4 float
#define DBL8 double
```

<table>
<thead>
<tr>
<th>Supported Compilers</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU C/C++ on SUN Solaris</td>
</tr>
<tr>
<td>GNU C/C++ on Linux</td>
</tr>
<tr>
<td>Borland C++ 5.02</td>
</tr>
<tr>
<td>Visual C++ 5.0</td>
</tr>
<tr>
<td>Syman tec C++ 7.0</td>
</tr>
<tr>
<td>F77 on SUN Solaris</td>
</tr>
<tr>
<td>G77 on Linux</td>
</tr>
<tr>
<td>Absoft Pro Fortran 6.0</td>
</tr>
</tbody>
</table>

7
#define INT8 __int64
#define DBL4 float
#define DBL8 double

#endif

/**** Borland C++ */
#ifdef __BORLANDC__

#define EXPORT __export

double dnan = 0.0/0.0, dinf = 1.0/0.0;

#define INT1 __int8
#define INT2 __int16
#define INT4 __int32
#define INT8 __int64
#define DBL4 float
#define DBL8 double

#endif

#ifdef __SC__

#define EXPORT __export

#include <fp.h>
double dnan = NAN, dinf = INFINITY;

#define INT1 char
#define INT2 short
#define INT4 int
// #define INT8 not supported by Symantec :(
#define DBL4 float
#define DBL8 double

#endif

/**** If none applies then set some defaults values */

#ifndef EXPORT

#define EXPORT

#define INT1 signed char
#define INT2 short
#define INT4 int
#define INT8 long
#define DBL4 float
#define DBL8 double

#endif

typedef void* voidp;
The first lines of the include file define the macro `EXTERN`. Every C/C++ compiler should define a macro called `_cplusplus` if the code is compiled as C++. If it is compiled as C this macro should not be defined.

The reason to define `EXTERN` as `extern "C"` is that C++ uses different internal names for the functions. E.g. if you compile the introductory example under Syman-tec C++ you will get a function name like `?sum@YAH@W0002Z` which is not easy to remember.

Then we start with the compiler dependend definitions. For example the `EXPORT` macro allows to export a function from a DLL. Under Unix (Gnu C/C++) you do not need to declare, but under Windows you may have private routines. Every C/C++ compiler generates a unique macro, e.g. `_GNUC_` the GNU C/C++ compiler or `_BORLANDC_` for the Borland C/C++ compiler. This macro is used to set the macro `EXPORT` correctly such that this function will be really exported. An alternative way under Windows would be to define a `vecsum.def` file, please have a look at your compiler manual.

Furthermore some mathematical constants and values are defined as well as the data types `DBL8` and so on.

### 3.4 Checking for exportable functions

Once you have compiled a DLL you might want to check which are the exportable functions and how they are named.

Under Windows you can use the Explorer:

1. go on `vecsum.dll`
2. use the right mouse button to get the context menu
3. choose quick view (under Windows 95/98 you might have to download and to install the quick view from Microsoft)
4. under the section Export Table you will find the exported functions

Under Unix you might use the tool `nm`. Just do `nm name_of_your_DLL.so` which will show all accessible functions from your shared library.

### 4 XploRe commands for using DLLs

XploRe has four commands which are to be used with DLLs:

#### 4.1 dlopen

dlopen opens (loads) a DLL. Call the DLL named `MyDLL` with

```c
h = dlopen("MyDLL")
```
or

\[ h = \text{dlopen("absolute\_path\_to\_MyDLL")} \]

In the first call, you need to have set the path to your DLL by \texttt{setenv}. Or alternatively, you can set the environment variable \texttt{XPL4DLL} with \texttt{setenv} to your DLL directory before starting XploRe.

The output from \texttt{dlopen}, the scalar \( h \), is the handle of the opened DLL. Each time you open a DLL, this handle is incremented by 1. The handle is used to distinguish between two opened DLLs with the same name.

### 4.2 \texttt{dlcall}

\texttt{dlcall} calls functions from an already loaded DLL. Call the function \texttt{MyFunction} with

\[ \text{dlcall("MyFunction", arg1, arg2, ...)} \]

or

\[ \text{dlcall(h, "MyFunction", arg1, arg2, ...)} \]

The second call applies, when you want to call explicitly the function \texttt{MyFunction}, which belongs to the DLL with handle \( h \). You can omit the handle, if there is only one \texttt{MyFunction} in all the opened DLLs.

The arguments \( \text{arg1, arg2, ...} \) are the XploRe objects (scalars, vectors, matrices, arrays) which are to be passed to the function \texttt{MyFunction}.

\textbf{Note:} \texttt{dlcall} is not used here, you will find an example in link!!!!!!

### 4.3 \texttt{dlcallex}

\texttt{dlcallex} calls functions from an already loaded DLL. Call the function \texttt{MyFunction} with

\[ \text{dlcallex("MyFunction", param, opt)} \]

or

\[ \text{dlcallex(h, "MyFunction", param, opt)} \]

The second call applies, when you want to call explicitly the function \texttt{MyFunction}, which belongs to the DLL with handle \( h \). You can omit the handle, if there is only one \texttt{MyFunction} in all the opened DLLs.

The argument \texttt{param} contains the XploRe objects (scalars, vectors, matrices, arrays) which are to be passed to the function \texttt{MyFunction}. The parameter \texttt{opt} tells how they passed to the C/C++ function.
4.4 dlquery

dlquery can be used to find out which DLLs are opened. It is invoked with

\[ q = \text{dlquery}() \]

In case, that no DLL is open, it simply gives the value 0. In the case, that one or more DLLs are open, dlquery returns a list with three components:

- **name**: name of DLL
- **location**: absolute path and name of the DLL
- **count**: how many times the DLL is opened

4.5 dlclose

dlclose closes (unloads) one or all DLLs. It is called by

\[ \text{dlclose("MyDLL")} \]

or

\[ \text{dlclose(h)} \]

or

\[ \text{dlclose()} \]

The first and second call close a specified DLL by its name or by its handle, respectively. The last call will close all open DLLs.