HOW TO GENERATE A DLL FROM A MATLAB FUNCTION SCRIPT TO RUN IN LABVIEW

1. Basic Requirements

1.1 Make sure you have:

- MATLAB compiler
- C/C++ compiler. Most people can afford to buy MS Visual C/C++ so why not try (eg: Open Watcom http://www.openwatcom.org/)

1.2 Make sure your MATLAB compiler and C/C++ compiler are running: Try mathworks technical note 1621 (http://www.mathworks.com/support/tech-notes/1600/1621.shtml). This note should give you enough information about how to setup your MATLAB compiler and C/C++ compiler.

2. A simple example:

The problem of generating a dll from MATLAB to work in LabVIEW is that the MATLAB compiler requires special data types (e.g. mxArray). It is therefore required to write a special 'wrapper function' which interfaces the MATLAB generated source code with a more 'standard' ANSI C code. The following example illustrates the procedure. The code following has been implemented using MATLAB 5.3, MATLAB Compiler 2.0, Open Watcom 1.0, and LabVIEW 6.1.

2.1 Write the following m-code function and save as foo.m:

```
function y = foo(x)
y = 2*x
```

2.2 Compile into 'C' code: Type the following command in the MATLAB prompt:

```
>>mcc -t -L C -W lib:foolib -h foo.m
```

This will generate the following files:

foo.c: contains the implementation of foo (Mfoo) and the interfacing functions (mlfFoo, mlxFoo)

foo.h: contains the prototypes of mlfFoo and mlxFoo

foolib.c: contains the implementations of foolibInitialize and foolibTermintate, necessary to initialize and terminate mlfFoo

foolib.h: contains the prototypes of mlxFoo, foolibInitialize and foolibTerminate.

Foolib.exports: contains the symbols to export in the dll.

2.3 Create foo_wrapper.c. This is the wrapper function which will allow the use of the C implementation of foo.m. The code is listed and commented below.

```
/* This file foo_wrapper.c */
#include "matlab.h"
#include "foodll.h"
#include "matrix.h"

//main wrapper function definition
double wrapper_main(double *in1){
   //declare variable to deliver result
   double out;

   //Create two pointers of mxArray type to store inputs and outputs
   mxArray *in1_ptr, *out1_ptr;

   //Allocate input pointer to a 1 by 1 double, real matrix
   in1_ptr = mxCreateDoubleMatrix(1,1,mxREAL);

   //Move the data from the input to the pointer
   fill(mxGetPr(in1_ptr),in1,1);

   //Initialise foo implementation
   foolibInitialize();

   //Pass values to mlfFoo and receive in mxArray type variable
   out1_ptr = mlfFoo(in1_ptr);

   //Terminate foo implementation
   foolibTerminate();

   //Move from mxArray type to double type
   fill(in1,mxGetPr(out1_ptr),1);

   //move data to output variable
   out = *in1;

   //Return value
   return(out);
}
void fill(double *out, double *in, int size){
   //This function moves data from one type to another
   int i;
   for(i=0;i<size;i++)
      out[i] = in[i];
}
```

2.4 Create foodll.h which contains prototypes of functions in foo_wrapper.c. The code is listed below.

```
//This file foodll.h
double wrapper_main(double *in1);
void fill(double *out, double *in, int size);
```

2.5 Add entry point to foolib.exports: add the following line to the file foolib.exports:

```
wrapper_main
```

2.6 Build using the following command in MATLAB:

```
>>mbuild –link shared foo_wrapper.c foo.c foolib.c foolib.exports
```

2.7 You have now generated a file called foo_wrapper.dll, which contains the function wrapper_main.c among others. Use this function making sure the input (arguments) and output (return value) are of 8-bit double type. Also, notice that you should pass a pointer to the value and not the value itself to the function.

3. Advanced examples

The following are more advanced examples which illustrate how to pass matrices from LabVIEW to a MATLAB function, and also when you have multiple m-files which need to be compiled and inserted in a DLL.

3.1 Multiplication of two matrices

We will follow a more systematic approach to the development of the DLL in MATLAB to interface with LabVIEW. This example has been developed using MATLAB ver. 6.5 R13 with MATLAB Compiler 3.0, LabVIEW 7.0 Express, and MSVC 6.0.

3.1.1 Write the following m-code function and save it as ab.m

```
function [c] = ab(a,b);

c=a*b;
```

3.1.2 Compile intro C code:

```
>>mcc –t –L C –W lib:ablib –h ab.m libmmfile.mlib
```

This generates similar files as the ones described in the first example, and will not be discussed here. The main point of

concern will be the passing of matrices from standard C to MATLAB mxArray and viceversa.

3.1.3 Create the wrapper function. To do so, you can either begin from scratch or use LabVIEW to help you write the prototype, so you avoid mistakes and have data-type compatibility. Please refer to LabVIEW bookshelf manual: ***Using External Code in LabVIEW***. Save the file as *atimesb.c*.

```
/* Call Library source file */

#include "matlab.h"
#include "matrix.h"
#include "ablib.h"

void atimesb(double a[], short int num_rows, short int num_cols, double
b[]);

void fill(double *out, double *in, int rows, int cols);

void atimesb(double a[], short int num_rows, short int num_cols, double
b[])
      {
    int i=0, j=0;

      // Create pointers for the two matrices
      mxArray *a_ptr, *b_ptr;

      // Assign pointer to adequate size matrices
      a_ptr = mxCreateDoubleMatrix(num_rows,num_cols,mxREAL);
      b_ptr = mxCreateDoubleMatrix(num_rows,num_cols,mxREAL);

    //fill the data in mxArray pointer
    fill(mxGetPr(a_ptr),a,num_cols,num_rows);
    fill(mxGetPr(b_ptr),b,num_cols,num_rows);


      //Initialise the matlab implementation
      ablibInitialize();

      //Pass values to mlfAb
      a_ptr=mlfAb(a_ptr, b_ptr);

      //Terminate the matlab implementation
      ablibTerminate();

      //Move data from mxArray type to output
      fill(a,mxGetPr(a_ptr),num_rows,num_cols);

      //Clean memory
      mxDestroyArray(a_ptr);
      mxDestroyArray(b_ptr);

}
```

```
void fill(double *out, double *in, int rows, int cols){
    int i=0, j=0;
    // Fill in the data
      for(i=0;i<rows;i++){
          for(j=0;j<cols;j++){
              out[(i*cols)+j] = in[(j*rows)+i];
          }
      }
}
```

ANSI C uses row-major storage, while MATLAB uses column-major storage. This basically implies that the data has to be stored in different order to be interpreted correctly. For example if in ANSI C a matrix is defined as ***A***, and the data is passed into the mxArray type in the same order, the MATLAB will interpret the data as $\boldsymbol{A}^T$, and viceversa. For more information on this please look at: ***MATLAB C Math Library*** manual, in the section of numeric arrays. Available at:

http://www.mathworks.com/access/helpdesk_r12p1/help/pdf_doc/mathlib/cmath_ug2b.pdf

3.1.4 Add function entry point to *ablib.exports*: Add the following line to this file:

```
atimesb
```

3.1.5 Finally, build the DLL by using the following command:

```
>>mbuild -link shared atimes.c ab.c ablib.c ablib.exports
```

3.2 Design of a predictive controller

I will refer very briefly to this example by pointing out only the differences with the other examples and listing only the important parts of the written code.

The DLL is generated from three files: *smacmpc.m*, *mpcdesign.m* and *constraintsdesign.m*.

The objective of this DLL is to design a Model Predictive Controller, that is calculate the matrices involved in the on-line optimization. This DLL, does not perform the optimization, just provides the matrices, given the system matrices (A, B, C, D), the constraints (P, p, F, f, E, and e), the prediction and control horizons (Hp, and Hu), and the diagonal of the cost weights (q and r).

The file *smacmpc.m* is the main point of entry, in which the other two functions are called.

The main feature of this example is that the all the functions return more than one argument. Please pay attention to the wrapper file to see how to handle this situation.

3.2.1 Compilation: The files must be compiled together to generate one object file and not several. Type the following in the MATLAB command line:

```
>>mcc -t -L C -g -W lib:mpclib -h smacmpc.m constraintsdesign.m
mpcdesign.m libmmfile.mlib
```

3.2.2 The use of the –g option allows for debugging in MSVC, though this and other options for debugging are not discussed here. Please refer to:

http://www.mathworks.com/support/solutions/data/27671.shtml

and the MATLAB Compiler for and IDE with MSVC which provides a fully integrated development environment.

3.2.3 The wrapper function: *mpcdll.c*. The important part here is how the multiple outputs of the mlfSmacmpc function are handled.

```
/* Call Library source file */
#include "matlab.h"
#include "matrix.h"
#include "mpclib.h"


        void mpc(double a[], double b[], double c[], double d[], short int n,
        short int m, short int l, short int n1, short int n2, short int n3,
        double Hp, double Hu, double qu[], double ru[], double P[], double p[],
        double F[], double f[], double E[], double e[], double psi[], double theta[],
        double tau[], double Sr[], double Sq[], double W[], double w1[], double w2[],
        double w3[]);


void fill(double *out, double *in, int rows, int cols);


void mpc(double a[], double b[], double c[], double d[], short int n,
        short int m, short int l, short int n1, short int n2, short int n3,
        double Hp, double Hu, double qu[], double ru[], double P[], double p[],
        double F[], double f[], double E[], double e[], double psi[], double theta[],
        double tau[], double Sr[], double Sq[], double W[], double w1[], double w2[],
        double w3[]){

        /* This explains the arguments of this function:
        a, b, c and d are the system matrices over which the predictor is built
        n, m  and l are the order, number of inputs and number of outputs respectively */


        // Create pointers
```

```
        mxArray *a_ptr, *b_ptr, *c_ptr, *d_ptr;
        mxArray *qu_ptr, *ru_ptr, *hp_ptr, *hu_ptr;
        mxArray *psi_ptr, *theta_ptr, *tau_ptr;
        mxArray *Sr_ptr, *Sq_ptr;
        mxArray *P_ptr, *p_ptr, *F_ptr, *f_ptr, *E_ptr, *e_ptr;
        mxArray *W_ptr, *w1_ptr, *w2_ptr, *w3_ptr;


        // Assign pointer to adequate size matrices
        a_ptr = mxCreateDoubleMatrix(n,n,mxREAL);
        b_ptr = mxCreateDoubleMatrix(n,m,mxREAL);
        c_ptr = mxCreateDoubleMatrix(l,n,mxREAL);
        d_ptr = mxCreateDoubleMatrix(l,m,mxREAL);

        P_ptr = mxCreateDoubleMatrix(n1,l,mxREAL);
        p_ptr = mxCreateDoubleMatrix(n1,1,mxREAL);
        F_ptr = mxCreateDoubleMatrix(n2,m,mxREAL);
        f_ptr = mxCreateDoubleMatrix(n2,1,mxREAL);
        E_ptr = mxCreateDoubleMatrix(n3,m,mxREAL);
        e_ptr = mxCreateDoubleMatrix(n3,1,mxREAL);

        qu_ptr = mxCreateDoubleMatrix(l,1,mxREAL);
        ru_ptr = mxCreateDoubleMatrix(m,1,mxREAL);

        hp_ptr = mxCreateDoubleScalar(Hp);
        hu_ptr = mxCreateDoubleScalar(Hu);

        psi_ptr = mxCreateDoubleMatrix(l*Hp,n,mxREAL);
        theta_ptr = mxCreateDoubleMatrix(l*Hp,m*Hu,mxREAL);
        tau_ptr = mxCreateDoubleMatrix(l*Hp,m,mxREAL);

        Sr_ptr = mxCreateDoubleMatrix(m*Hu,m*Hu,mxREAL);
        Sq_ptr = mxCreateDoubleMatrix(l*Hp,l*Hp,mxREAL);

        W_ptr = mxCreateDoubleMatrix(n1*Hp+n2*Hu+n3*Hu,m*Hu,mxREAL);
        w1_ptr = mxCreateDoubleMatrix(n1*Hp+n2*Hu+n3*Hu,m,mxREAL);
        w2_ptr = mxCreateDoubleMatrix(n1*Hp+n2*Hu+n3*Hu,n,mxREAL);
        w3_ptr = mxCreateDoubleMatrix(n1*Hp+n2*Hu+n3*Hu,1,mxREAL);


        //fill the data in mxArray pointer
    fill(mxGetPr(a_ptr),a,n,n);
    fill(mxGetPr(b_ptr),b,m,n);
    fill(mxGetPr(c_ptr),c,n,l);
    fill(mxGetPr(d_ptr),d,m,l);

    fill(mxGetPr(P_ptr),P,l,n1);
    fill(mxGetPr(p_ptr),p,1,n1);
    fill(mxGetPr(F_ptr),F,m,n2);
    fill(mxGetPr(f_ptr),f,1,n2);
    fill(mxGetPr(E_ptr),E,m,n3);
    fill(mxGetPr(e_ptr),e,1,n3);

    fill(mxGetPr(qu_ptr),qu,1,l);
    fill(mxGetPr(ru_ptr),ru,1,m);

        //matlab implementation
    mpclibInitialize();
    psi_ptr =
mlfSmacmpc(&theta_ptr,&tau_ptr,&Sr_ptr,&Sq_ptr,&W_ptr,&w1_ptr,&w2_ptr,&w3_ptr,a_ptr,b_ptr
,c_ptr,d_ptr,hp_ptr,hu_ptr,qu_ptr,ru_ptr,P_ptr,p_ptr,F_ptr,f_ptr,E_ptr,e_ptr);
    mpclibTerminate();

        //Move data from mxArray type to output
        fill(psi,mxGetPr(psi_ptr),l*Hp,n);
    fill(theta,mxGetPr(theta_ptr),l*Hp,m*Hu);
    fill(tau,mxGetPr(tau_ptr),l*Hp,m);
    fill(Sr,mxGetPr(Sr_ptr),m*Hu,m*Hu);
    fill(Sq,mxGetPr(Sq_ptr),l*Hp,l*Hp);

    fill(W,mxGetPr(W_ptr),n1*Hp+n2*Hu+n3*Hu,m*Hu);
```

```
    fill(w1,mxGetPr(w1_ptr),n1*Hp+n2*Hu+n3*Hu,m);
    fill(w2,mxGetPr(w2_ptr),n1*Hp+n2*Hu+n3*Hu,n);
    fill(w3,mxGetPr(w3_ptr),n1*Hp+n2*Hu+n3*Hu,1);

    mxDestroyArray(a_ptr);
    mxDestroyArray(b_ptr);
    mxDestroyArray(c_ptr);
    mxDestroyArray(d_ptr);
    mxDestroyArray(qu_ptr);
    mxDestroyArray(ru_ptr);
    mxDestroyArray(hp_ptr);
    mxDestroyArray(hu_ptr);
    mxDestroyArray(psi_ptr);
    mxDestroyArray(theta_ptr);
    mxDestroyArray(tau_ptr);
    mxDestroyArray(Sr_ptr);
    mxDestroyArray(Sq_ptr);
    mxDestroyArray(P_ptr);
    mxDestroyArray(p_ptr);
    mxDestroyArray(F_ptr);
    mxDestroyArray(f_ptr);
    mxDestroyArray(E_ptr);
    mxDestroyArray(e_ptr);
    mxDestroyArray(W_ptr);
    mxDestroyArray(w1_ptr);
    mxDestroyArray(w2_ptr);
    mxDestroyArray(w3_ptr);

        }


void fill(double *out, double *in, int rows, int cols){
    int i=0, j=0;
    // Fill in the data
        for(i=0;i<rows;i++){
            for(j=0;j<cols;j++){
                out[(i*cols)+j] = in[(j*rows)+i];
            }
        }
}
```

3.2.4 Finally add *mpc* to the *mpclib.exports* file.

4. Additional information can be found in:

4.1 http://www.mathworks.com/support/solutions/data/25176.shtml

4.2 http://www.mathworks.com/support/solutions/data/23932.shtml

4.3 http://www.mathworks.com/support/solutions/data/27671.shtml

4.4 http://zone.ni.com/devzone/conceptd.nsf/webmain/5CF9A9FFD774028586256869005FF2ED?opendocument