How to get Real Time Data into Matlab

First make sure you have Visual Studio 6.0 installed. You're going to have to build a mex file in visual studio. A mex file is just C code that has been compiled to a library (a dll for windows) for use within Matlab. Start a new project; the type should be dll. Select *Project Settings* from the menu. Choose tab *C/C++*. Under *Preprocessor Definitions,* add *MATLAB_MEX_FILE.* Add a file with extension .def to your project, and enter only the following lines:

```
LIBRARY xxxx.dll
EXPORTS mexFunction
```

The file xxxx.dll is the mex file you wish to build. The name of this file will be the name of the function you call in Matlab. Now add a .cpp file to the project with the following header:

```
#include "c:\Program Files\Matlab7\extern\include\mex.h"
```

Make sure to change the directory to point to your mex.h file. It should also be possible to set the directory as a project option. The last step for making a mex file is to write the following function:

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[])
{
}
```

This is the function Matlab executes when you run the mex file. The type mxArray is a Matlab data type that stores vectors and matrices. The parameters nlhs and nrhs are the number of arguments the function should return and the number of arguments the function accepts respectively. The remaining parameters are the input and output arguments themselves.

The next step is to write the Real Time Data Exchange code (RTDX). Before we look at the code for the host, there is some code that needs to be written for the target.

```
if(RTDX_isOutputEnabled(&ochan)) {
        short a[ORD],i;

        for(i=0; i<POLYORD; i+=2) {
                a[i] = (short)x0[i>>1].real;
                a[i+1] = (short)x0[i>>1].imag;
        }
        if(RTDX_writing == NULL)
                RTDX_write(&ochan,a,sizeof(short)*ORD);
}
```

This snippet checks that the output channel is enabled and writes ORD/2 complex valued numbers to the RTDX log file. Usually you will want execute RTDX_write after processing each buffer. However, keep in mind that Matlab is slow. Depending on your machine and how much computation your .m file does, you should expect to be able to read RTDX data just 1 to 20 times per second. This means either you may need to write data less often or you will need to figure out a way to always read the newest data in the buffer. Otherwise, the data you read will be noticeably delayed.

Also on the target, add:

RTDX_CreateOutputChannel(ochan);

outside of any functions. And place:

```
TARGET_INITIALIZE();
RTDX_enableOutput(&ochan);
```

in your initialization routine. Finally you need the header files:

```
#include "rtdx.h"
#include "target.h"
```

Let's return to the host and fill in the mex function.

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[])
{
    mxArray *param, *out;
    double p, *outDBL;

    param=prhs[0];
    p = mxGetScalar(param); //(A)
    if(p==0) {
        out = mxCreateDoubleMatrix(1,1,mxREAL); //(B)
        outDBL = mxGetPr(out); //(C)
        *outDBL = (double)open(); //(D)
    } else if(p<0) //(E)
       close();
    else { //(F)
       out = mxCreateDoubleMatrix(p,1,mxREAL);
       outDBL = mxGetPr(out);
       get(outDBL,p);
    }

    if(p>=0) { //(G)
        if(nlhs>0)
                plhs[0]=out;
        else
                mxDestroyArray(out);
    }
}
```

The function assumes we are being passed one scalar. It is retrieved using mxGetScalar (A). If the value is zero, create a one-element vector (B). Get a pointer to the real part of the vector (these are doubles) (C). Open the RTDX channel and put the result in the vector we just made. If the value is negative (E), close the stream. Otherwise (F), create a vector of length p and read the results into it. Finally, if p is positive (G), return the results to Matlab; destroy the vector otherwise.

Below is the open function:

```
int open()
{
        long status;
        HRESULT hr;
        CBoardProcessor boardproc;
        TCHAR BoardName[MAX_PATH];
        TCHAR ProcessorName[MAX_PATH];

        ::CoInitialize(NULL);
        if (boardproc.GetBoard((TCHAR*)&BoardName) == FALSE) {
                cerr << _T("\n*** Error: Unable to get desired
board!.\n***");
                return -1;
        }
        if
(boardproc.GetProcessor((TCHAR*)&BoardName,(TCHAR*)&ProcessorName) ==
FALSE) {
                cerr << _T("\n*** Error: Unable to get desired
processor!.\n***");
                return -1;
        }
        hr = rtdx.CreateInstance(__uuidof(RTDXINTLib::RtdxExp));
        if (FAILED(hr))
                return hr;
        status = rtdx-
>SetProcessor(T2BSTR(BoardName),T2BSTR(ProcessorName));
        printf("status %ld (1)\n",status);
        if (status!=Success)
                 return -1;
        status = rtdx->ConfigureRtdx(1,1024,4);
        printf("status %ld (2)\n",status);
        if(status!=Success)
                 return -1;
        status = rtdx->EnableRtdx();
        printf("status %ld (3)\n",status);
        if(status!=Success)
                 return -1;
        status = rtdx->Open("ochan","r");
        printf("status %ld (4)\n",status);
        if(status!=Success)
                 return -1;
        return 1;
}
```

This will open ochan for reading. You will need to include the file boardprocessor.h and add boardprocessor.c to the project. These can be found in C:\CCStudio\Examples\hostapps\rtdx\eventprog\. This file has been written to ask for the board and processor. My machine is setup with only one board and processor; so, I just commented out the input lines and set the appropriate variables to zero. If your setup is different, you may need to do more work. Finally, the close routine is simple enough:

```
void close()
{
        rtdx->DisableRtdx();
        rtdx->Close();
}
```

Be careful when opening and closing. If you close the stream twice, the dll will display a message box with an error. If you open the channel twice, there seems to be more serious problems. Thus, it’s a good idea to always close the channel before opening it.

And finally the real work:

```
void get(double *out, int num)
{
        long status=0;
        short val;
        int i=0;

        for(i=0; i<num; i++)
                out[i]=-1;
        i=0;
        while(i<num) {
                status = rtdx->ReadI2(&val); //(A)
                if(status==Success)
                        out[i++]=val;
                else {
                        return;
                }
        }
}
```

This function reads num two-byte values into the out array (A). If at any time there is an error, the function returns. The remaining unread values will be -1.

Add the following to the file:

```
#include <stdio.h>
#include "boardprocessor.h"

#import "c:\CCStudio\cc\bin\rtdxint.dll"
using namespace RTDXINTLib;


#define Success 0x0
#define EOLF 0x80030002
#define FAIL 0x80004005
#define WARNING 0x80004004
#define NODATA 0x8003001E

IRtdxExpPtr rtdx;
void get(double *out, int num);
int open();
void close();
```

Make sure you get the path right for rtdxint.dll. You can check Code Composer help to see what some of the other status message are which have been defined here.

The last step is to build the project. You will need to either add the directory of the dll to your Matlab path or change to that directory from within Matlab. At any time you can clear the dll from Matlab's memory using clear xxxx.dll. (Remember that any static and global variables will remain in memory until you clear the file.) Supposing you name the file readRTDX.dll, you can now use the following in Matlab to read in 10 RTDX values:

```
%load the dll and open RTDX
if readRTDX(0)==1
        %read 10 shorts (doubles in Matlab)
        vals=readRTDX(10),
        %close
        readRTDX(-1),
end
```

Todd Goldfinger