USB-2416 CSharp (C#) Temperature Voltage Continuous Read Example

The following program continuously acquires data from the USB-2416. Before the acquisition is starts it further check to make sure that InstaCal was used to configure the channels for temperature and voltage - see comment in code. Modifying the Device string to 2408 this program should work with the USB-2408 as it is of the same hardware design.

To access the MccDaq API, add a reference to the MccDaq object. Adding the reference is usually accomplished by right clicking the Project [under the Project Explorer] and selecting Add Reference. The complete project is attached to the article and was created with Visual Studio 2008 using CSharp. Compiling and execution is quick as there is minimal Windows overhead and associated code. To recreate the project use Visual Studio's File->New Project, select Visual C# and select Console Application. Create the new project and add the reference to the MccDaq Dot Net component. The easiest way to do this is to right mouse click the project in the Project Explorer and choose Add Reference. Select MccDaq from the NET list. Of course this assumes that you have installed the InstaCal software.

Although not shown below, a few convenience functions were added such as IsError, GetBoardNum, DisplayData, CreateFileHeader and WaitForKey. To view all the functions download the complete Visual Studio 2008 project by extracting the zip file at the bottom.

The IsError function checks the error number in the ErrorInfo object and if not zero displays the error message. The GetBoardNum function searches for a device matching the identifying string and when found exits returning the board number assigned by InstaCal. The board number is used to get the controlling device object that has AInScan function. The CreateFileHeader writes acquisition information to the beginning of the output file. The Display data writes the data to the file and console screen. And WaitForKey does just that - waits for someone to press the spacebar.

Disclaimer:

The attached Code or Example is provided As Is. It has not been tested or validated as a product, for use in a deployed application or system, or for use in hazardous environments. You assume all risks for use of the Code or Example.

```
//////////////////////////////////////////////////////////////////////////
using System;
using System.Runtime.InteropServices;
using MccDaq;//Project must have reference to MccDaq
using System.IO;

namespace TemperatureVoltageReadBackground
{
    class Program
    {
        public const int BLOCKSIZE = 50; //for best results use increments of packet size
which is 32
        public const int CHANCOUNT = 4;
        public const int FIRSTCHANNEL = 0;
        public const int LASTCHANNEL = 3;
        public const int SAMPLERATE= 100;
        public const int BUFFERSIZE = BLOCKSIZE * CHANCOUNT;
        public const int HALFBUFFSIZE = BUFFERSIZE / 2;
        public const string DEVICE = "2416";

        //external function not found in MccDaq
        //Used to determine InstaCal setup
        //************* Compile to x86 for 32 bit function call
****************************
        [DllImport("cbw32.dll")]
        public extern static long cbGetConfig([MarshalAs(UnmanagedType.I4)] int InfoType,
                                    [MarshalAs(UnmanagedType.I4)] int BoardNum,
                                    [MarshalAs(UnmanagedType.I4)] int DevNum,
                                    [MarshalAs(UnmanagedType.I4)] int ConfigItem,
                                    [MarshalAs(UnmanagedType.I8)] ref long ConfigVal
                                   );


//**********************************************************************************************
        public static StreamWriter fStream;


        static void Main(string[] args)
        {
            //Project must have reference to MccDaq
            MccDaq.ErrorInfo RetVal;
            int BoardNum = 0;
            int InfoType = 2;
            int DevNum = 0;
            int ConfigItem = 235; //Item number to get channel type
            long ConfigVal = 0;
            long ret;
            int Rate = SAMPLERATE;

            BoardNum = GetBoardNum(DEVICE);
```

```
if (BoardNum == -1)
{
   Console.WriteLine("No USB-{0} detected!", DEVICE);
   WaitForKey();
   return;
}

MccBoard daq = new MccDaq.MccBoard(BoardNum);

IntPtr buffer = MccService.ScaledWinBufAllocEx(BUFFERSIZE);

if (buffer == IntPtr.Zero)
{
   Console.WriteLine("Bad Memory Handle");
   WaitForKey();
   return;
}

short[] chType = new short[CHANCOUNT];
short[] chArray = new short[CHANCOUNT];
Range[] chRange = new Range[CHANCOUNT];

//channels I want to read
chArray[0] = 0;
chArray[1] = 1;
chArray[2] = 2;
chArray[3] = 3;


//Use InstaCal and set the first two channels to thermocouples
//and the second two to voltage. Set the Data Rate to 1000
//When using lengthy thermocouples set the channel Data Rate to 60Hz and reduce
//the program's sample rate
chType[0] = 2;//type for thermocouple
chType[1] = 2;
chType[2] = 5;//type for voltage
chType[3] = 5;

//Check to make sure the user ran InstaCal and set the channels appropriately
for (DevNum = 0; DevNum < CHANCOUNT; DevNum++)
{
   ret = cbGetConfig(InfoType, BoardNum, DevNum, ConfigItem, ref ConfigVal);
```

```
          if (ConfigVal != chType[DevNum])
          {
             Console.Write("Configuration Error!\nRun InstaCal\n" +
                       "\tset channels 0 - 3 to temperature\n");
             WaitForKey();
             return;
          }
        }

        chRange[0] = Range.Bip10Volts;
        chRange[1] = Range.Bip10Volts;
        chRange[2] = Range.Bip10Volts;
        chRange[3] = Range.Bip10Volts;

        RetVal = daq.ALoadQueue(chArray, chRange, CHANCOUNT);
        IsError(RetVal);

        //ALoadQueue overrides channels and range in AInScan
        RetVal = daq.AInScan(   FIRSTCHANNEL,
                       LASTCHANNEL,
                       BUFFERSIZE,
                       ref Rate,
                       Range.Bip10Volts,
                       buffer,
                       ScanOptions.Background | ScanOptions.ScaleData |
ScanOptions.Continuous
                     );
        IsError(RetVal);
        fStream = new StreamWriter(@"C:\Users\Public\Documents\DataFile1608G.asc");
        CreateFileHeaders(chArray); //writes basic info to the beginning of the file

        int Count = 0;
        int Index = 0;
        short daqStatus;
        bool ReadLower = true;
        double[] theArray = new double[BUFFERSIZE];

        System.ConsoleKeyInfo cki = new System.ConsoleKeyInfo();

        //Loop until key press
        do
        {
                RetVal = daq.GetStatus(out daqStatus, out Count, out Index,
FunctionType.AiFunction);
                if ((Index >= HALFBUFFSIZE) & ReadLower) //check for 50% more data
                {
                    //get lower half of buffer - ScaledWinBufToArray returns engineering units
                    RetVal = MccService.ScaledWinBufToArray(buffer, theArray, 0,
HALFBUFFSIZE);
                    IsError(RetVal);

                    DisplayData(theArray, HALFBUFFSIZE / CHANCOUNT);
                    ReadLower = false; //flag that controls the next read
                }
                else if ((Index < HALFBUFFSIZE) & !ReadLower)
                {
                    //get the upper half  - ScaledWinBufToArray returns engineering units
                    RetVal = MccService.ScaledWinBufToArray(buffer, theArray, HALFBUFFSIZE,
HALFBUFFSIZE);
                    IsError(RetVal);

                    DisplayData(theArray, HALFBUFFSIZE / CHANCOUNT);
                    ReadLower = true;//flag that controls the next read
                }

            } while (!Console.KeyAvailable);

            cki = Console.ReadKey();

            //flush any buffered data out to disk
            fStream.Close();

            //stop the  acquisition
            RetVal = daq.StopBackground(FunctionType.AiFunction);

            //free up memory
            MccService.WinBufFreeEx(buffer);

            WaitForKey();


        }
```