Biometrics Ltd

Application Programmable Interface (API) Examples

RM84592-1

API Examples

1. Microsoft Visual Basic 2005 Interface Example for Windows
2. Microsoft Visual Basic 6.0 Interface Example for Windows
3. Microsoft Visual C++ 2005 or 2010 Interface Example for Windows
4. Microsoft Visual C++ 2005 or 2010 Waveform Plotting Example for Windows
5. Microsoft Visual C++ 6 Interface Example for Windows
6. Microsoft C Console Application Interface Example for Windows
7. Microsoft Visual C# Waveform Plotting Example for Windows
8. MATLAB Interface

The functions that allow other applications to interface to the Biometrics Analysis/Management Software are contained in the file OnLineInterface.dll. **This DLL is registered with Windows during installation** and afterwards allows direct data and control access to the Analysis/Management Software.

Two interface functions are provided - the first provides data and control status information with start/stop control and the second gets the next block of data from a channel buffer. Both of these functions are provided with on-line help and a number of supporting constants.

Note that these functions are members of the **OnLine** Class and an object of that class must be defined before using the functions.

In order to collect data from a DataLOG/DataLINK/DataLITE, the Biometrics Analysis/Management Software must **not be in File Save mode**. The section covering **How to Transfer Data to Other Applications** gives more information.

Many of the examples provided and listed below make use of a DataLINK installation. If a DataLOG or DataLITE installation is being used, all references to the **'Biometrics DataLINK Interface Library'** must use the path to the DataLOG/DataLITE directory; for example, C:\\Program Files\\Biometrics Ltd\\DataLog\\OnLineInterface.dll. Of course, using search and replace within the example code, all references to **DataLINK** may simply be changed to **DataLOG** or **DataLITE**.

OnLineStatus Function

C++ Prototype:	int __stdcall OnLineStatus(long channel, long statusType, long *pStatus);
VB Prototype:	Sub OnLineStatus(channel As Long, statusType As Long, pStatus As Long);

This function provides data and control status information with start/stop control.

The statusType parameter must be one of the StatusType command constants listed below. Note that any command will return ONLINE_COMMSFAIL in pStatus if communications with the hardware fails.

StatusType Constant	**Value**	**Comment**
ONLINE_GETERROR	0	channel is unused. Return with the current status in pStatus.
ONLINE_GETENABLE	1	pStatus returns with 1 if the specified channel is enabled or 0 if it is not.
ONLINE_GETRATE	2	pStatus returns with the number of samples per second on the specified channel.
ONLINE_GETSAMPLES	3	pStatus returns with the number of unread samples on the specified channel or ONLINE_OVERRUN.
ONLINE_GETVALUE	4	pStatus returns with the (current value + 4000) on the specified channel.
ONLINE_START	5	channel is unused. Start or re-start the data transfer.
ONLINE_STOP	6	channel is unused. Stop the data transfer.

Error Return Constants	Value	Comment
ONLINE_OK	0	No communications or buffer errors.
ONLINE_COMMSFAIL	-3	Communications with the hardware has failed.
ONLINE_OVERRUN	-4	The internal buffer has overflowed and some data has been lost.

Note that samples returned by this function using ONLINE_GETSAMPLES will only change at a very slow rate of around once per second. To increase this rate, use ONLINE_START to begin data transfer before using ONLINE_GETSAMPLES repeatedly.

If more than one DataLOG or DataLINK DLK900 unit is being used simultaneously to collect data then the channel number needs to reflect which unit is being accessed. Each unit has 9 channels of data (8 analogue and 1 digital) which are accessed in sequence. The channel number is therefore:

(unit number * 9) + channel number within the unit

where the unit number can be 0 to 7 and the channel number 0 to 8. The unit number is its order in the Detected Units Window with unit 0 being at the top. Note that units marked without Transfer or with Removed are still counted.

If a recording is made using DataLITE, The channel number is

(Record Window * 9) + channel number within the Record Window

OnLineGetData Function

C++ Prototype:	int __stdcall OnLineGetData(long channel, long sizeMsToRead, SAFEARRAY **DataArray, long *pActualSamples);
VB Prototype:	Sub OnLineGetData(channel As Long, sizeMsToRead As Long, DataArray() As Integer, pActualSamples As Long);

This function return the next block of data received on the specified channel. Analogue channels are returned when channel is 0 to 7; channel 8 returns the digital values as a set of bits similar to those defined in Data Format.

The number of samples returned in the DataArray array will be no more than (sizeMsToRead * sample rate) where sizeMsToRead is a number of milliseconds. If insufficient samples are available then all that is available will be returned.

If no error occurred, pActualSamples will return with the actual number of samples returned in the DataArray array. If an error did occur, pActualSamples will return with a negative value.

OnLineGetData requires a SAFEARRAY containing short integers to return the data. This can be either created by the user (possibly using SafeArrayCreateVector(VT_I2, 0, samplesRequested)) or the exact size array may be created by OnLineGetData itself. The following Microsoft C++ example illustrates passing the address of a zero dimension array to force OnLineGetData to create a suitable size SAFEARRAY and return with its address.

If more than one DataLOG or DataLINK DLK900 unit is being used simultaneously to collect data then the channel number needs to reflect which unit is being accessed. Each unit has 9 channels of data (8 analogue and 1 digital) which are accessed in sequence. The channel number is therefore:

(unit number * 9) + channel number within the unit

where the unit number can be 0 to 7 and the channel number 0 to 8. The unit number is its order in the Detected Units Window with unit 0 being at the top. Note that units marked without Transfer or with Removed are still counted. See the Microsoft Visual C++ 2005 Waveform Plotting Example for Windows for an illustration of multiple units.

If a recording is made using DataLITE, The channel number is

(Record Window * 9) + channel number within the Record Window

```
// create a zero dimension array;
SAFEARRAY *pSafeArrayZeroDim = SafeArrayCreate(VT_I2, 0, 0);
// this will cause the DLL to create an array using
// SafeArrayCreateVector(VT_I2, 0, samplesRequested)
SAFEARRAY *pSafeArray = pSafeArrayZeroDim;
// read up to 100mS of data as values -4000 to +4000
long sampleNum;
if ((pOnline->OnLineGetData(channel, 100, &pSafeArray, &sampleNum) == ONLINE_OK) &&
       (sampleNum > 0))
{
    short *pValue;
    if (SafeArrayAccessData(pSafeArray, (LPVOID *)&pValue) == S_OK)
    {
        // now do a simple plot of the data; -4000 at bottom and +4000 at the top
    }
}
SafeArrayDestroy(pSafeArray);   // clean up
SafeArrayDestroy(pSafeArrayZeroDim);
```

Doing part of this in Visual Basic:

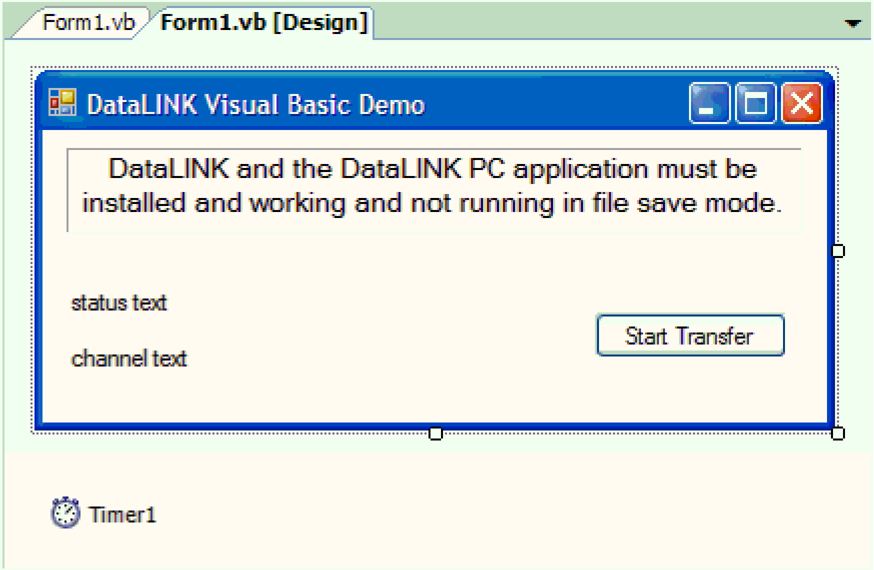
```
Dim status, samples As Integer
Dim a() As Short

objInterface.OnLineGetData(0, 100, a, samples)
If samples > 0 Then
    ' :
End If
```

1. Microsoft Visual Basic 2005 Interface Example for Windows

To use these functions within **Microsoft Visual Basic**, the interface reference must be enabled for the current project (select **Add References** from the **Project** menu). Once the **'Biometrics DataLINK Interface Library'** is enabled, use the **Object Browser** with **OnLineInterfaceLib** selected to see the available functions and constants along with on-line help about the functions.

Click here to download project files.

Layout the following form and add two labels named **StatusText** and **ChannelText**. Also, add a button named **StartStopButton** and a 1000mS interval timer called **Timer1**.



DataLINK and the DataLINK PC application must be installed and working and not running in file save mode.

```
Public Class Form1
    ' DataLINK/DataLOG and the PC application must be installed
    ' and working and not running in file save mode.
    ' Declare an interface object to provide a link to the data
    Private DataLINK As OnLineInterfaceLib.OnLine
    Dim started As Boolean
    Private Sub Timer1_Tick(ByVal sender As System.Object,
                            ByVal e As System.EventArgs) Handles Timer1.Tick
        ' Here every second
        Dim status As Integer        ' Declare variables to use in data transfer
        Dim samples As Integer
        Dim channel As Integer
        Dim data() As System.Int16
        Dim y As Integer
        Dim Point As Integer
        ' Get some status information and display it
        channel = 0     ' demo uses channel 1 only for simplicity
        DataLINK.OnLineStatus(channel, OnLineInterfaceLib.StatusOption.ONLINE_GETERROR,
status)
        If status = OnLineInterfaceLib.ErrorCode.ONLINE_OK Then
            DataLINK.OnLineStatus(channel,
OnLineInterfaceLib.StatusOption.ONLINE_GETSAMPLES, samples)
            ' Display info on the form
            StatusText.Text = 'status OK, ' & samples & ' samples in buffer for channel 1'
            ' Get up to 1000mS of data from channel 0 and read each value
            ' variable data is not assigned values as it is used to receive the
            ' data - this will produce a compiler warning
            DataLINK.OnLineGetData(channel, 1000, data, samples)
            For Point = 0 To samples - 1
                y = data(Point) ' Demo just reads data but does nothing with it
            Next
        Else
            ' Display message if any problems communicating with DataLINK
            StatusText.Text = 'DataLINK Offline'
        End If
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object,
                            ByVal e As System.EventArgs) Handles MyBase.Load
        DataLINK = CreateObject('OnLine.Interface')
        started = False
    End Sub

    Private Sub StartStop_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
                                Handles StartStopButton.Click
        Dim status As Integer
        Dim enable As Integer
        Dim channel As Integer
        Dim samplesPerSec As Integer
        channel = 0     ' demo uses channel 1 only for simplicity
        If started Then
            ' Here if stop button pressed so try to stop and
            ' set button text to "Start Transfer"
            DataLINK.OnLineStatus(channel, OnLineInterfaceLib.StatusOption.ONLINE_STOP,
status)
            If status = OnLineInterfaceLib.ErrorCode.ONLINE_OK Then
                StartStopButton.Text = 'Start Transfer'
                ChannelText.Text = ''
                started = False
            End If
        Else
            ' Here if start button pressed so try to start,
            ' set button text to 'Stop Transfer' and display the sampling rate
            DataLINK.OnLineStatus(channel, OnLineInterfaceLib.StatusOption.ONLINE_START,
status)
            If status = OnLineInterfaceLib.ErrorCode.ONLINE_OK Then
                StartStopButton.Text = 'Stop Transfer'
                DataLINK.OnLineStatus(channel,
OnLineInterfaceLib.StatusOption.ONLINE_GETENABLE, enable)
                If enable Then
                    DataLINK.OnLineStatus(channel,
OnLineInterfaceLib.StatusOption.ONLINE_GETRATE, samplesPerSec)
                    ChannelText.Text = 'Channel 1: ' & samplesPerSec & ' samples per
second'
                Else
                    ChannelText.Text = 'Channel 1 disabled'
                End If
                started = True
            End If
        End If
    End Sub
End Class
```

2. Microsoft Visual Basic 6.0 Interface Example for Windows

To use these functions within **Microsoft Visual Basic**, the interface reference must be enabled for the current project (for example using VB6.0, select **References** from the **Project** menu). Once the **'Biometrics DataLINK Interface Library'** is enabled, use the **Object Browser** with **OnLineInterface.Lib** selected to see the available functions and constants along with on-line help about the functions. Note that these functions are members of the **OnLine** Class and an object of that class must be defined before using the functions.

Click here to download project files.

```
' Declare an interface object to provide a link to the data
Private objInterface As OnLine

' Declare a few variables to use in the data transfer
Dim status As Long
Dim samples As Long
Dim channel As Long
Dim data() As Integer
Dim y As Integer, Point As Integer

' Get some status information and display it
objInterface.OnLineStatus 0, ONLINE_GETERROR, status
objInterface.OnLineStatus 0, ONLINE_GETSAMPLES, samples
Debug.Print 'status: ' & status & ', ' & samples & ' samples in buffer'

' Get up to 100mS of data from channel 0 and read each value
channel = 0
objInterface.OnLineGetData channel, 100, data(), samples
For Point = 0 To samples - 1
    y = data(Point)
Next
```

3. Microsoft Visual C++ 2005 or 2010 Interface Example for Windows

The example code below assumes a simple Wizard created MFC single-document application called VClinkTestNET. No other header or library files are required except that the OnLineInterface.dll must have been correctly installed as part of the normal installation process.

Click here to download project files.

It is assumed that the installation directory is **C:\Program Files\Biometrics Ltd\DataLINK.** If a DataLOG is used rather than a DataLINK, use search and replace within the following code to change all **DataLINK** references to **DataLOG**. In order to collect data from a DataLOG/DataLINK/DataLITE, the Biometrics Analysis/Management Software must **not be in File Save mode**.

Use the class wizard to add the ExitInstance() function override; this will add to VClinkTestNET.h and VClinkTestNET.cpp. The code specified in bold will need to be entered (or copied from this help).

In VClinkTestNET.h

```
public:
    virtual int ExitInstance();
```

In VClinkTestNET.cpp

```
// Get information from the DLL about its functionality.
// Place this line outside of any functions before using the DLL.
#import 'C:\\Program Files\\Biometrics Ltd\\DataLINK\\OnLineInterface.dll' no_namespace
IOnLine *pOnline; // Declare a global pointer to the interface class instance

BOOL CVClinkTestNETApp::InitInstance()
{
    ::
    AfxEnableControlContainer();
    // To use the interface class, create an instance:
    CoCreateInstance(__uuidof(OnLine), NULL, CLSCTX_INPROC_SERVER,
                     __uuidof(IOnLine), (void**)&pOnline);
    if (pOnline == NULL)
    {
        AfxMessageBox(_T('Cannot find OnLineInterface.dll.\n\n')
            _T('Please ensure that DataLINK was correctly\n')
            _T('installed from the original CD as this also\n')
            _T('registers the DLL with Windows.'));
        return FALSE;
    }
    ::
}

int CVClinkTestNETApp::ExitInstance()
{
    if (pOnline != NULL)
        pOnline->Release(); // release the interface class instance
    return CWinApp::ExitInstance();
}
```

Again, use the class wizard to add the WM_TIMER and WM_DESTROY message handling functions and the OnInitialUpdate() function override; this will add to VClinkTestNETView.h and VClinkTestNETView.cpp.

In VClinkTestNETView.h

```
public:
    afx_msg void OnTimer(UINT_PTR nIDEvent);
public:
    afx_msg void OnDestroy();
public:
    virtual void OnInitialUpdate();
```

In VClinkTestNETView.cpp

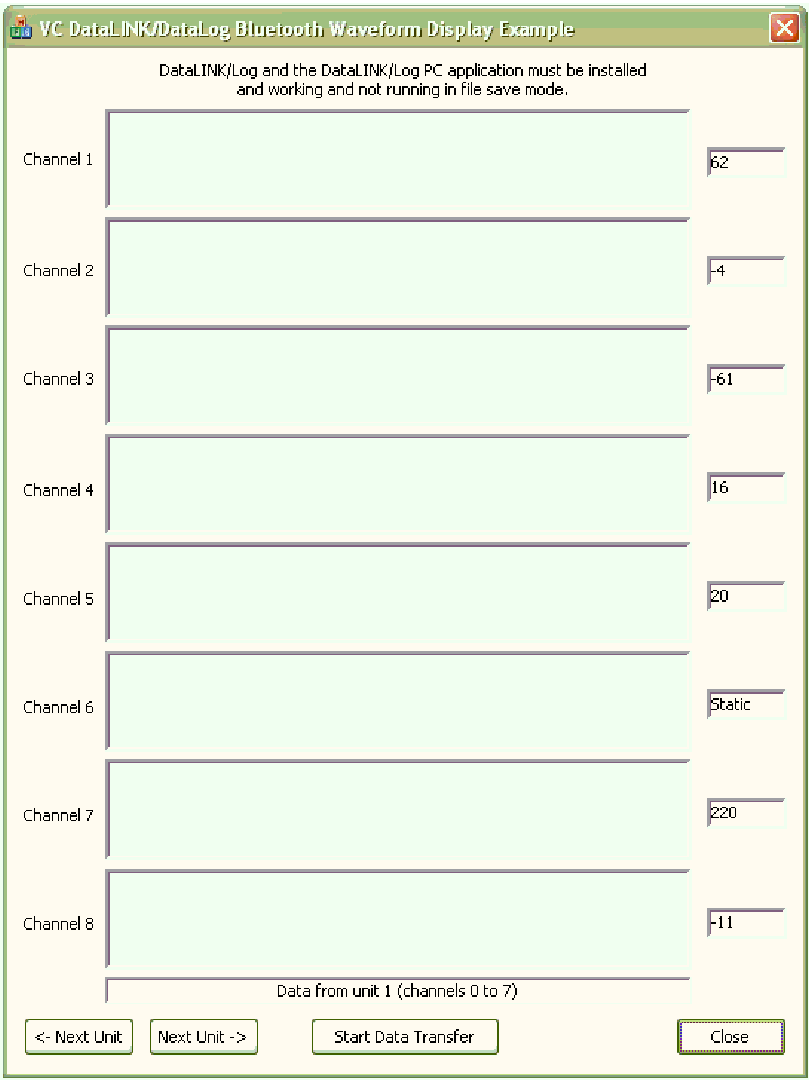
```
// the following 2 lines must be outside of any functions before using the DLL.
#import 'C:\\Program Files\\Biometrics Ltd\\DataLINK\\OnLineInterface.dll' no_namespace
extern IOnLine *pOnline;
long g_status;

BEGIN_MESSAGE_MAP(CVClinkTestNETView, CView)
// Standard printing commands
    ::
    ON_WM_TIMER()
    ON_WM_DESTROY()
END_MESSAGE_MAP()

void CVClinkTestView::OnDraw(CDC* pDC)
{
    // For example, display the sample rate used on channel 0.
    if (g_status < 0)
    {
        pDC->TextOut(0, 0, _T('Cannot communicate with DataLINK!\n')
            _T('Is DataLINK running and connected to the DataLINK hardware?'));
        return;
    }
    CString text;
    text.Format(_T('rate = %ld'), g_status);
    pDC->TextOut(0, 0, text);
}

#define ID_TIMER_VIEW 1 // almost any number will do!

void CVClinkTestNETView::OnInitialUpdate()
{
    CView::OnInitialUpdate();

    // TODO: Add your specialized code here and/or call the base class
    // generate a ID_TIMER_VIEW timer event every 100mS
    SetTimer(ID_TIMER_VIEW, 100, NULL);
}

void CVClinkTestNETView::OnDestroy()
{
    CView::OnDestroy();

    // TODO: Add your message handler code here
    KillTimer(ID_TIMER_VIEW);
}

void CVClinkTestNETView::OnTimer(UINT_PTR nIDEvent)
{
    // TODO: Add your message handler code here and/or call default
    // here 10 times per second
    if (nIDEvent == ID_TIMER_VIEW)
    {
        static long statusLast = -99; // a value that is never returned by OnLineStatus and
                                      // will therefore force an initial redraw of the view
        pOnline->OnLineStatus(0, ONLINE_GETRATE, &g_status);
        if (g_status != statusLast)
        {
            // a simple redraw of the complete view whenever the
            // contents of g_status changes
            GetDocument()->UpdateAllViews(NULL);
            statusLast = g_status; // redraw only once
        }
    }
    CView::OnTimer(nIDEvent);
}
```

4. Microsoft Visual C++ 2005 or 2010 Waveform Plotting Example for Windows

The example code below assumes a simple Wizard created MFC dialog based application called VClinkWaveform. No other header or library files are required except that the OnLineInterface.dll must have been correctly installed as part of the normal installation process.

The dialog resource contains 8 picture controls, color white and type rectangle, type with ID's IDC_WAVEFORM_CH1 to IDC_WAVEFORM_CH8 and 8 text controls with ID's IDC_VALUE_CH1 to IDC_VALUE_CH8. These controls have their 'Client Edge' and 'Sunken' properties set for appearance sake.

To demonstrate using more than one DataLOG or DataLINK unit, two buttons have been added with IDs IDC_NEXT_DOWN and IDC_NEXT_UP and a static text control with ID IDC_UNIT_TITLE. A button with ID IDC_STARTSTOP has also been added along with some simple descriptive text boxes as shown:



VC DataLINK/DataLog Bluetooth Waveform Display Example

DataLINK/Log and the DataLINK/Log PC application must be installed
and working and not running in file save mode.

Channel 1 62

Channel 2 -4

Channel 3 -61

Channel 4 16

Channel 5 20

Channel 6 Static

Channel 7 220

Channel 8 -11

Data from unit 1 (channels 0 to 7)

Click here to download project files.

It is assumed that the installation directory is **C:\Program Files\Biometrics Ltd\DataLINK.** If a DataLOG is used rather than a DataLINK, use search and replace within the following code to change all **DataLINK** references to **DataLOG**. In order to collect data from a DataLOG/DataLINK/DataLITE, the Biometrics Analysis/Management Software must **not be in File Save mode**.

Use the class wizard to add the ExitInstance() virtual function (function override) to the VClinkWaveformApp class. The code specified in bold will need to be entered (or copied from this help).

In VClinkWaveformApp.h

```
public:
     virtual int ExitInstance();
```

In VClinkWaveformApp.cpp

```
// Get information from the DLL about its functionality.
// Place this line outside of any functions before using the DLL.
#import 'C:\\Program Files\\Biometrics Ltd\\DataLINK\\OnLineInterface.dll' no_namespace
IOnLine *pOnline; // Declare a global pointer to the interface class instance

BOOL VClinkWaveformApp::InitInstance()
{
    // Initialize OLE libraries
    if (!AfxOleInit())
    {
        AfxMessageBox(_T('OLE init failed'));
        return FALSE;
    }

    AfxEnableControlContainer();
    // To use the interface class, create an instance:
    CoCreateInstance(__uuidof(OnLine), NULL, CLSCTX_INPROC_SERVER,
                     __uuidof(IOnLine), (void**)&pOnline);
    if (pOnline == NULL)
    {
        AfxMessageBox(_T('Cannot find OnLineInterface.dll.\n\n')
            _T('Please ensure that DataLINK was correctly\n')
            _T('installed from the original CD as this also\n')
            _T('registers the DLL with Windows.'));
        return FALSE;
    }
    ::
}

int VClinkWaveformApp::ExitInstance()
{
    if (pOnline != NULL)
        pOnline->Release(); // release the interface class instance
    return CWinApp::ExitInstance();
}
```

Again, use the class wizard to add the WM_TIMER and WM_DESTROY message handler functions, the IDC_NEXT_DOWN, IDC_NEXT_UP and IDC_STARTSTOP command handler functions for the BN_CLICKED message to the VClinkWaveformDlg class; this will add to VClinkWaveformDlg.h and VClinkWaveformDlg.cpp. The code specified in bold will need to be entered (or copied from this help).

In VClinkWaveformDlg.h

```
public:
    bool m_dataTransfer;            // true when transferring data
    CRect m_waveformRect[8];        // drawing rectangles
    CPoint m_waveformPointLast[8];  // the last point plotted for each channel
    int m_OnLineChannelOffset;      // unit offset of data in the DLL

    afx_msg void OnTimer(UINT_PTR nIDEvent);
    afx_msg void OnDestroy();
    afx_msg void OnBnClickedStartStop();
    afx_msg void OnBnClickedNextDown();
    afx_msg void OnBnClickedNextUp();
```

In VClinkWaveformDlg.cpp

```
// the following 2 lines must be outside of any functions before using the DLL.
#import 'C:\\Program Files\\Biometrics Ltd\\DataLINK\\OnLineInterface.dll' no_namespace
extern IOnLine *pOnline;
#define ID_TIMER_VCLINK  1      // value not critical
```

```
int IDvalues[8] = { IDC_VALUE_CH1, IDC_VALUE_CH2,
    IDC_VALUE_CH3, IDC_VALUE_CH4,
    IDC_VALUE_CH5, IDC_VALUE_CH6,
    IDC_VALUE_CH7, IDC_VALUE_CH8 };
int IDwaveform[8] = { IDC_WAVEFORM_CH1, IDC_WAVEFORM_CH2,
    IDC_WAVEFORM_CH3, IDC_WAVEFORM_CH4,
    IDC_WAVEFORM_CH5, IDC_WAVEFORM_CH6,
    IDC_WAVEFORM_CH7, IDC_WAVEFORM_CH8 };

BEGIN_MESSAGE_MAP(CVClinkTestNETView, CView)
// Standard printing commands
        ::
    ON_WM_TIMER()
    ON_WM_DESTROY()
    ON_BN_CLICKED(IDC_STARTSTOP, &CVClinkWaveformDlg::OnBnClickedStartStop)
    ON_BN_CLICKED(IDC_NEXT_DOWN, &CVClinkWaveformDlg::OnBnClickedNextDown)
    ON_BN_CLICKED(IDC_NEXT_UP, &CVClinkWaveformDlg::OnBnClickedNextUp)
END_MESSAGE_MAP()

BOOL CVClinkWaveformDlg::OnInitDialog()
{
    :
    pOnline->OnLineStatus(0, ONLINE_STOP, NULL); // just to make sure
    m_OnLineChannelOffset = 0;
    m_dataTransfer = false;

    CPoint DlgOrg(0, 0);
    ClientToScreen(&DlgOrg);
    for (int channel = 0; channel < 8; channel++)
    {
        CWnd *pWaveform = GetDlgItem(IDwaveform[channel]);
        pWaveform->GetClientRect(m_waveformRect[channel]); // get size of wave rect

        // adjust trace rectangle coordinates to be relative to the dialog box and not the
wave box.
        CPoint WaveformOrg(0, 0);
        pWaveform->ClientToScreen(&WaveformOrg);
        m_waveformRect[channel].MoveToXY(WaveformOrg.x-DlgOrg.x, WaveformOrg.y-DlgOrg.y);
        // initialise plotting to start in middle of LHS
        m_waveformPointLast[channel].x = m_waveformRect[channel].left;
        m_waveformPointLast[channel].y = (m_waveformRect[channel].top +
        m_waveformRect[channel].bottom) / 2;
    }

    SetTimer(ID_TIMER_VCLINK, 50, NULL); // waveform update uses this timer

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CVClinkWaveformDlg::OnBnClickedNextDown()
{
    if (m_OnLineChannelOffset != 0)
    m_OnLineChannelOffset -= 9;  // skip 8 analogue and 1 digital channel
    CString s;
    s.Format(_T('Data from unit %d (channels %d to %d)'),
             (m_OnLineChannelOffset / 9) + 1, m_OnLineChannelOffset,
             m_OnLineChannelOffset + 7);
    SetDlgItemText(IDC_UNIT_TITLE, s);
}

void CVClinkWaveformDlg::OnBnClickedNextUp()
{
    if (m_OnLineChannelOffset != (7 * 9)) // max of 8 units having 9 channels each
    m_OnLineChannelOffset += 9;  // skip 8 analogue and 1 digital channel
    CString s;
    s.Format(_T('Data from unit %d (channels %d to %d)'),
            (m_OnLineChannelOffset / 9) + 1, m_OnLineChannelOffset,
            m_OnLineChannelOffset + 7);
    SetDlgItemText(IDC_UNIT_TITLE, s);
}

void CVClinkWaveformDlg::OnBnClickedStartStop()
{
    if (m_dataTransfer)
    {
        pOnline->OnLineStatus(0, ONLINE_STOP, NULL);
        SetDlgItemText(IDC_STARTSTOP, _T('Start Data Transfer'));
        m_dataTransfer = false;
    }
    else
    {
        pOnline->OnLineStatus(0, ONLINE_START, NULL);
        SetDlgItemText(IDC_STARTSTOP, _T('Stop Data Transfer'));
        m_dataTransfer = true;
    }
}

void CVClinkWaveformDlg::OnDestroy()
{
    CDialog::OnDestroy();

    KillTimer(ID_TIMER_VCLINK);
}

void CVClinkWaveformDlg::OnTimer(UINT_PTR nIDEvent)
{
    // Here every 50mS. Update values in the dialogue
    static long valueLast[8];
    if (nIDEvent == ID_TIMER_VCLINK)
    {
        for (int channel = 0; channel < 8; channel++)
        {
            long valueNew;
            pOnline->OnLineStatus(m_OnLineChannelOffset + channel, ONLINE_GETVALUE,
&valueNew);
            if (valueNew != valueLast[channel])
            {
                // change of state screen update to reduce flicker
                SetDlgItemInt(IDvalues[channel], valueNew - 4000, TRUE);
                valueLast[channel] = valueNew;
            }
            if (m_dataTransfer)
            {
                long sampleNum;
                SAFEARRAY *pSafeArrayZeroDim = SafeArrayCreate(VT_I2, 0, 0); // create a
zero dim array
                // this will cause the DLL to use SafeArrayCreateVector(VT_I2, 0,
samplesRequested)
                SAFEARRAY *pSafeArray = pSafeArrayZeroDim;
                // read up to 1000mS of data as values -4000 to +4000
                if ((pOnline->OnLineGetData(m_OnLineChannelOffset + channel, 1000,
                    &pSafeArray, &sampleNum) == ONLINE_OK) && (sampleNum > 0))
                {
                    short *pValue;
                    CDC *pDC = GetDC();
                    if (pDC && SafeArrayAccessData(pSafeArray, (LPVOID *)&pValue) == S_OK)
                    {
                        // now do a simple plot of the data; -4000 at bottom and +4000 at
the top
                        CRect *pRect = &m_waveformRect[channel];
                        CPoint *pPoint = &m_waveformPointLast[channel];
                        int height = pRect->Height() - 1;
                        for (int sample = 0; sample < sampleNum; sample++, pValue++)
                        {
                            if (pPoint->x >= (pRect->right - 1))
                            {
                                // erase waveform when plotting reaches RHS
                                pDC->FillSolidRect(*pRect, RGB(255, 255, 255));
                                pPoint->x = pRect->left;
                            }
                            pDC->MoveTo(*pPoint);
                            pPoint->y = pRect->bottom - ((*pValue + 4000) * height / 8000)
- 1;
                            pPoint->x++;
                            pDC->LineTo(*pPoint);
                            pDC->SetPixel(*pPoint, RGB(0, 0, 0)); // LineTo does not plot
last point
                        }
                    }
                    ReleaseDC(pDC);
                }
                SafeArrayDestroy(pSafeArray);  // clean up
                SafeArrayDestroy(pSafeArrayZeroDim);
            }
        }
    }
    CDialog::OnTimer(nIDEvent);
}
```

5. Microsoft Visual C++ 6 Interface Example for Windows

The example code below assumes a simple Wizard created MFC multi-document application called VClinkTest. No other header or library files are required except that the OnLineInterface.dll must have been correctly installed as part of the normal installation process.

Click here to download project files.

It is assumed that the installation directory is **C:\Program Files\Biometrics Ltd\DataLINK.**
If a DataLOG is used rather than a DataLINK, use search and replace within the following code to change all **DataLINK** references to **DataLOG**. In order to collect data from a DataLOG/DataLINK/DataLITE, the Biometrics Analysis/Management Software must **not be in File Save mode**.

In VClinkTest.cpp

```
// Get information from the DLL about its functionality.
// Place this line outside of any functions before using the DLL.
#import 'C:\\Program Files\\Biometrics Ltd\\DataLINK\\OnLineInterface.dll' no_namespace
IOnLine *pOnline; // Declare a global pointer to the interface class instance

BOOL CVClinkTestApp::InitInstance()
{
    ::
    AfxOleInit(); // Ensure that OLE has been initialised (required)
    // To use the interface class, create an instance:
    CoCreateInstance(__uuidof(OnLine), NULL, CLSCTX_INPROC_SERVER,
                     __uuidof(IOnLine), (void**)&pOnline);
    if (pOnline == NULL)
    {
        AfxMessageBox('Cannot find OnLineInterface.dll.\n\n'
            'Please ensure that DataLINK was correctly\n'
            'installed from the original CD as this also\n'
            'registers the DLL with Windows.');
        return FALSE;
    }
    ::
}

int CVClinkTestApp::ExitInstance()
{
    if (pOnline != NULL)
        pOnline->Release(); // release the interface class instance
    return CWinApp::ExitInstance();
}
```

In VClinkTestView.cpp

```
// the following 2 lines must be outside of any functions before using the DLL.
#import 'C:\\Program Files\\Biometrics Ltd\\DataLINK\\OnLineInterface.dll' no_namespace
extern IOnLine *pOnline;

void CVClinkTestView::OnDraw(CDC* pDC)
{
    // For example, display the sample rate used on channel 0.
    // Note that this example this will only update the text when the screen is re-drawn.
    long status;
    pOnline->OnLineStatus(0, ONLINE_GETRATE, &status);
    if (status < 0)
    {
        pDC->TextOut(0, 0,
            'Cannot communicate with DataLINK!  Is DataLINK '
            'running and connected to the DataLINK hardware?');
        return;
    }
    CString text;
    text.Format('rate = %ld', status);
    pDC->TextOut(0, 0, text);
}
```

6. Microsoft C Console Application Interface Example for Windows

The example code below assumes a simple Console application. No other header or library files are required except that the OnLineInterface.dll must have been correctly installed as part of the normal installation process.

Click here to download project files.

It is assumed that the installation directory is **C:\Program Files\Biometrics Ltd\DataLINK.** If a DataLOG is used rather than a DataLINK, use search and replace within the following code to change all **DataLINK** references to **DataLOG**. In order to collect data from a DataLOG/DataLINK/DataLITE, the Biometrics Analysis/Management Software must **not be in File Save mode**.

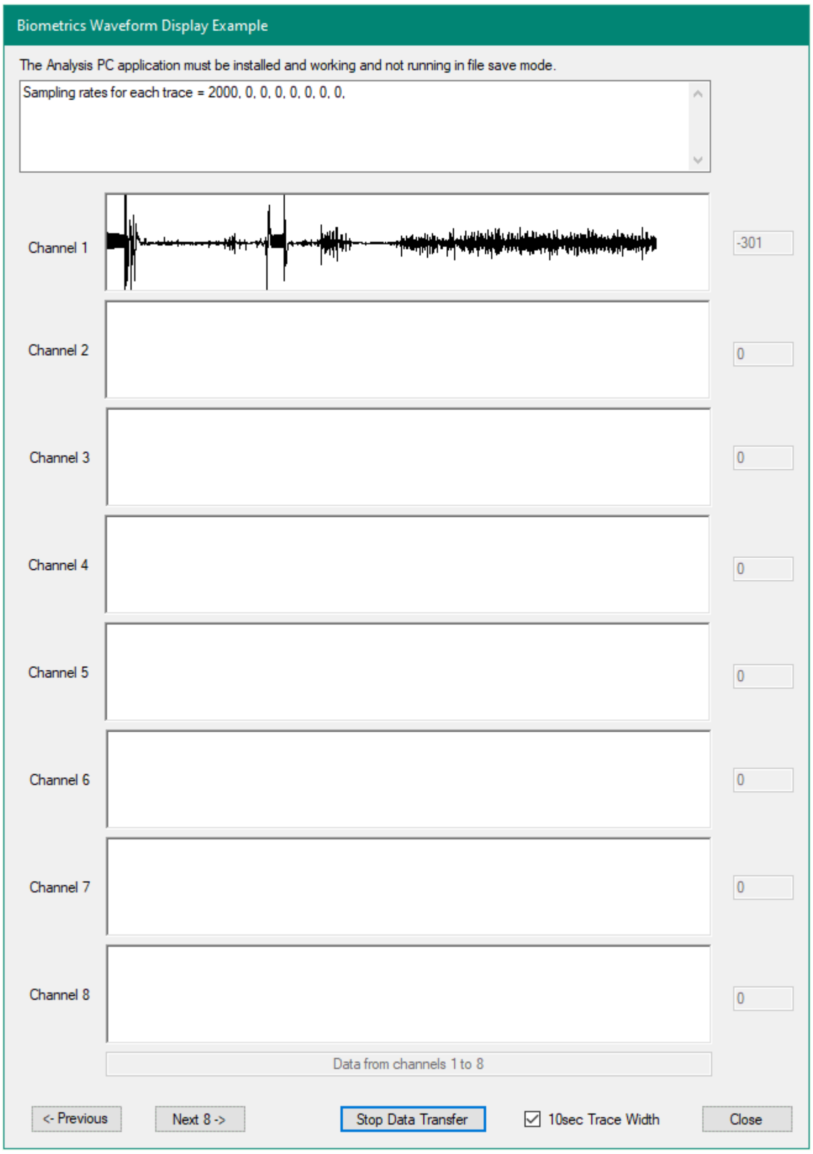
```
#include 'stdafx.h' // Assumes Microsoft Visual C++ environment

// Get information from the DLL about its functionality.
// Place this line outside of any functions before using the DLL.
#import 'C:\\Program Files\\Biometrics Ltd\\DataLINK\\OnLineInterface.dll' no_namespace
IOnLine *pOnline; // Declare a global pointer to the interface class instance

int main(int argc, char * argv[])
{
    // Ensure that OLE has been initialised (required)
    if (OleInitialize(NULL) != S_OK)
        return 2;
    // To use the interface class, create an instance:
    CoCreateInstance(__uuidof(OnLine), NULL, CLSCTX_INPROC_SERVER,
                     __uuidof(IOnLine), (void**)&pOnline);
    if (pOnline == NULL)
    {
        printf('Cannot find OnLineInterface.dll.\n');
        OleUninitialize();
        return 1;
    }

    long g_status=-99;
    pOnline->OnLineStatus(0, ONLINE_GETRATE, &g_status);
    printf('Channel 1 sample rate = %ld.\n', g_status);

    pOnline->Release(); // release the interface class instance
    OleUninitialize();
    return 0;
}
```

7. Microsoft Visual C# Waveform Plotting Example for Windows

This example uses Microsoft Visual Studio 2019 (also tested on 2015) to create a simple C# application that plots up to 8 waveforms in real-time from a DataLITE system. Although a DataLITE system is used in this example, a DataLOG or DataLINK system can also be used.

The example code below assumes a simple Visual C# 'Windows Desktop' - 'Windows Forms Application' created by 'Add New Project' from the File menu; the project name is WindowsFormsCsharpApp. No other header or library files are required except that the OnLineInterface.dll must have been correctly installed as part of the normal installation process.

The C# code included is not optimised in any way and there may be better ways of performing the same actions within C#. It is presented as operational code files that may be used as an introduction to the Biometrics interface.



Biometrics Waveform Display Example

The Analysis PC application must be installed and working and not running in file save mode.

Sampling rates for each trace = 2000, 0, 0, 0, 0, 0, 0, 0,

Channel 1: -301

Channel 2: 0

Channel 3: 0

Channel 4: 0

Channel 5: 0

Channel 6: 0

Channel 7: 0

Channel 8: 0

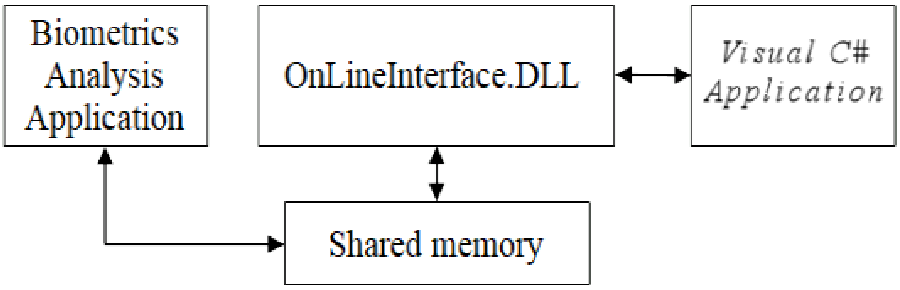
Data from channels 1 to 8

☑ 10sec Trace Width

There is an information pane at the top, 8 waveform panes below with current value displays for each. The DataLITE channel range may be changed using 2 buttons and a simple start/stop recording button is provided. The waveform panes can display either 10 seconds of data or 1 horizontal pixel per sample.

Before getting started...

The Biometrics Analysis application is responsible for gathering the data from the Biometrics hardware. From an external program's point of view, this application creates a shared block of memory into which the data is stored. When *OnLineInterface.DLL* is loaded, it looks for this block of memory and provides two functions that can be used to interface with it and hence the Analysis application. By loading *OnLineInterface.DLL* , this C# example application is able to use 2 functions from the DLL to access data and control how the Biometrics Analysis program operates.



There are some important points to remember.

- The Biometrics Application MUST be launched before the C# example application loads the DLL in order to create the shared memory.
- If the Biometrics Application is subsequently closed and reopened, it will see the existing shared memory and immediately close again.
- In order to collect data into shared memory and make it accessible to the example application, the Biometrics Application MUST have file save mode turned off (☒ showing on the toolbar) and transfer started (by pressing the ✓ icon on the toolbar (or by a ONLINE_START status function call from C#).
- Data is collected from the beginning of a start transfer into circular buffers that are a maximum of 50,000 samples in length for each channel. Data is removed by the C# application from the beginning of the buffer and added by Analysis to the end. So long as the C# application removes the data fast enough so that no more than 50,000 data values remain untaken from the buffer, no data is lost.
- Data is retained in the buffers after a stop transfer and this may be accessed by the C# application until a new recording is started. It is important that a buffer is not allowed to overflow because subsequent requests for data will return ONLINE_OVERRUN and no data.

Application Code

Click here to download project files.

The basic code structure is created by the new project Wizard. Changes have been made to the Form1.cs file and of course the Form1.Designer.cs file. The contents of Form1.cs is listed in parts below:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using System.Runtime.InteropServices;  // ******* addition for Biometrics Example *******
```

The InteropServices have been added because the DLL is a COM library. This DLL interface complicates the C# code because normally C# does not allow pointers to be specified. The DLL uses pointers and SafeArray, neither of which are easily supported by C# managed code.

```
namespace WindowsFormsCsharpApp
{
    public partial class Form1 : Form
    {
        // ******* Start of an addition for Biometrics Example *******
        public const string pathToDLL = "C:\\Program Files (x86)\\Biometrics
Ltd\\DataLITE\\OnLineInterface.dll";
        [DllImport(pathToDLL, EntryPoint = "_OnLineStatus@12", CallingConvention =
CallingConvention.StdCall)]
        //static extern unsafe OnLineError OnLineStatus(int channel, OnLineStatusType
statusType, int* pStatus);
        static extern OnLineError OnLineStatus(int channel, OnLineStatusType statusType,
ref int pStatus);

        [DllImport(pathToDLL, EntryPoint = "_OnLineGetData@16", CallingConvention =
CallingConvention.StdCall)]
        static extern unsafe OnLineError OnLineGetData(int channel, int sizeMsToRead,
SAFEARRAY** ppSafeArray, ref int pActualSamples);
```

The const string **pathToDLL must point to the location of the Biometrics DLL.**

The two DLL methods are then defined; note the use of the 'unsafe' keyword for the OnLineGetData method. Because this method (coded in C++) uses a pointer to a pointer, some code needs to be marked 'unsafe'.

Note that 'unsafe' does not mean that the code is unsafe to use! The way to view unsafe code is as unmanaged code that appears within a managed environment.

```
internal enum OnLineStatusType {
            ONLINE_GETERROR, ONLINE_GETENABLE, ONLINE_GETRATE, ONLINE_GETSAMPLES,
ONLINE_GETVALUE, ONLINE_START, ONLINE_STOP };

        internal enum OnLineError {
            ONLINE_COMMSFAIL = -3, ONLINE_OVERRUN = -4, ONLINE_OK = 0 };
```

For convenience, two groups of constants used by the DLL are defined as enum types.

```
const int TRACENUM = 8;
        const int BUFFERSIZE = 1000; // max size of data buffer in this application
        private int[] sampleRate = new int[TRACENUM];
        private int[] valueLast = new int[TRACENUM];
        private Point[] tracePointLast = new Point[TRACENUM];
        private double[] PixelsPerSample = new double[TRACENUM];
        private double[] xLast = new double[TRACENUM];

        private int OnLineChannelOffset = 0;
        private bool dataTransfer = false;
        private bool gotAnalysis = true;
        public CheckState checkBox10secCheckStateLast;
```

The first 2 constants define how many traces are in this application and the amount of storage required by the buffer used to hold the data from the DLL. Because data is obtained from the DLL using pointers, an 'unsafe' buffer is defined below to be passed to the DLL. This will be copied to a managed array for processing.

The variables above are as follows:

sampleRate An array holding the number of samples per second for each trace.

valueLast An array of values that are used to detect when a channel value displayed in the boxes on the right hand side changes.

tracePointLast An array of X/Y values containing the last point plotted in each trace box so that the next execution of the timer knows where to start drawing the next part of the waveform.

PixelsPerSample An array of values containing how many pixels correspond to each sample plotted when the '10 sec Trace width' button is pressed. These would normally be much less than 1.

xLast An array of accurate floating point x positions corresponding to the X value in tracePointLast. This will be truncated to an integer to set the X screen position for line drawing.

OnLineChannelOffset This variable keeps track of the channel number for the first trace (Channel 1).

dataTransfer Set and cleared when the start/Stop button is pressed.

gotAnalysis Used to detect when the analysis DLL is found or lost so that a single error message can be given.

checkBox10secCheckStateLast This is used to detect a change made to the 10 sec checkbox state in order to clear the waveform display ready for a new x axis scale.

```
private unsafe struct BUFFER
        {   // data for the DLL COM interface SAFEARRAY
            public fixed short data[BUFFERSIZE];
        }

        private unsafe struct SAFEARRAY
        {   // the DLL COM interface uses this
            public short cDims;
            public short fFeatures;
            public int cbElements;    // note that a long is 4 bytes in C++ and 8 bytes in 
C#; an int is 4 byes in both C++ and C#
            public int cLocks;
            public short *pData;
            public int cElements;
            public int lLbound;
        }
```

Two structures are used to mimic the operation of the C++ SAFEARRAY; these encapsulate a 1 dimensional data array of 2-byte integers.

Getting the data

```
private unsafe int GetData(int channel, int sizeMsToRead, short[] data)
        {
            BUFFER buf;
            if ((sizeMsToRead * sampleRate[channel % 8] / 1000) > BUFFERSIZE)
            {
                textBox1.Text += "make BUFFERSIZE larger?\r\n";
                return (int)OnLineError.ONLINE_OVERRUN;    // make BUFFERSIZE larger?
            }
            SAFEARRAY safeArray;
            safeArray.cDims = 1;
            safeArray.fFeatures = 0x2080;
            safeArray.cbElements = 2;
            safeArray.cLocks = 0;
            safeArray.pData = buf.data;
            safeArray.cElements = 1000;
            safeArray.lLbound = 0;
            SAFEARRAY* pSafeArray = &safeArray;
            // read up to 1000mS of data as values -4000 to +4000
            int sampleNum = 0;
            OnLineError error = OnLineGetData(channel, sizeMsToRead, &pSafeArray, ref 
sampleNum);
            if ((error == OnLineError.ONLINE_OK) && (sampleNum > 0))
                // note that if sampleNum = ONLINE_OVERRUN, data has been lost and 
ONLINE_STOP may be needed
                Marshal.Copy((IntPtr)buf.data, data, 0, data.Length);
            return sampleNum;
        }
```

The GetData() method is used to get the next block of samples from the DLL and place them into a managed array.

As mentioned earlier, the SAFEARRAY structure is filled and a pointer to a pointer of this structure is passed to OnLineGetData(). On return, this data is copied from the 'unsafe' array into a managed array.

```
private void InitialiseAllTraces()
        {
            for (int trace = 0; trace < TRACENUM; trace++)
                tracePointLast[trace].X = -1;    // initialise plotting to start in middle 
of LHS
        }
        // ******* End of this addition for Biometrics Example *******

        public Form1()
        {
            InitializeComponent();

            // ******* Start of an addition for Biometrics Example *******
            int status = 0;
            OnLineStatus(0, OnLineStatusType.ONLINE_STOP, ref status);    // just to make 
sure
            InitialiseAllTraces();
            for (int trace = 0; trace < TRACENUM; trace++)
                sampleRate[trace] = 0;
            checkBox10sec.CheckState = CheckState.Checked;
            // ******* End of this addition for Biometrics Example *******


        }
```

The above InitialiseAllTraces() method sets the X value of each member of tracePointLast[] to a value that will cause a clear trace to be executed at the correct moment in the timer method below.

The Form1 constructor method was added by the project wizard to be executed when the dialogue is created. Some initialisation takes place here including a command to stop any recording that may be in progress by the Biometrics Application.

```
private void timer1_Tick(object sender, EventArgs e)
        {
            // ******* Start of an addition for Biometrics Example *******
            // here every 10mS
            OnLineError error;
            bool sampleRateChanged = false;
            bool clearAllTraces = (checkBox10sec.CheckState != 
checkBox10secCheckStateLast) ? true : false;
            for (int trace = 0; trace < TRACENUM; trace++)
            {
                Control[] textBoxValList = { textBoxVal1, textBoxVal2, textBoxVal3, 
textBoxVal4, textBoxVal5, textBoxVal6, textBoxVal7, textBoxVal8 };


                Control[] traceBoxList = { traceCh1, traceCh2, traceCh3, traceCh4, 
traceCh5, traceCh6, traceCh7, traceCh8 };
```

The timer1_Tick() method was added by Visual Studio when a timer was added to the form; the timer interval should be set to 10mS in the properties of the timer icon on the form so that the code inside this method is executed every 10mS.

At the start of the timer method, some variables are initialised and a for loop started that will update all 8 traces and value displays. The Control arrays simplify the repetitive code needed to work with 8 similar traces and values.

```
int sampleRateNew = 0;
                error = OnLineStatus(trace, OnLineStatusType.ONLINE_GETRATE, ref 
sampleRateNew);

                if ((error == OnLineError.ONLINE_OK) && (sampleRateNew != 
(int)OnLineError.ONLINE_COMMSFAIL))
                {
                    if (!gotAnalysis)
                    {
                        gotAnalysis = true;
                        sampleRateChanged = true;    // show sampling rates
                    }
                    if (sampleRateNew != sampleRate[trace])
                    {
                        sampleRate[trace] = sampleRateNew; // redraw only once
                        sampleRateChanged = true;
                        PixelsPerSample[trace] = (double)traceBoxList[trace].Width / 
(double)(sampleRateNew * 10); // assuming a 10 sec trace width

                    }
```

Every 10mS, the sampling rates for every channel are checked to look for changes and to calculate the PixelsPerSample array values.

This process also checks for the presence of the Analysis application and DLL and reports an error message later if not found.

```
int valueNew = 0;
                    error = OnLineStatus(OnLineChannelOffset + trace, 
OnLineStatusType.ONLINE_GETVALUE, ref valueNew);
                    if ((error == OnLineError.ONLINE_OK) && (valueNew != valueLast[trace]))
                    {    // change of state screen update to reduce flicker
                        int val = valueNew - 4000;
                        textBoxValList[trace].Text = val.ToString();
                        valueLast[trace] = valueNew;

                    }
```

The above code updates the value displayed in each channel value box. To avoid unnecessary updates, the value is only drawn if it has changed. Like many other parts of this program, this Change of State (COS) mechanism reduces screen flicker.

```
if (dataTransfer)
                    {
                        short[] data = new short[1000];
                        int sampleNum = GetData(OnLineChannelOffset + trace, 100, data);
                        if (sampleNum == (int)OnLineError.ONLINE_OVERRUN)
                        {
                            int status = 0;
                            OnLineStatus(0, OnLineStatusType.ONLINE_STOP, ref status);
                            buttonStartStop.Text = "Start Data Transfer";
                            dataTransfer = false;

                        }
```

The above code snippet gets the latest set of samples as described earlier. Although 100mS of data is requested, most of the time the last 10mS will be obtained. Note that the managed array called 'data' needs to be large enough for 100mS of samples from the channel with the highest sampling rate.

The number of samples actually received by GetData() may be set to -3 (ONLINE_OVERRUN) if the circular buffers used by the DLL fill up and result in data loss. The only practical way out of this data loss is to stop the recording.

```
else if (sampleNum > 0)
                        {
                            Control traceBox = traceBoxList[trace];

                            Graphics g = traceBox.CreateGraphics();    // Create a Graphics
object for the Control.
                            // now do a simple plot of the data; -4000 at bottom and +4000
at the top
                            Pen blackPen = new Pen(Color.Black, 1);
                            Point p = tracePointLast[trace];
                            for (int sample = 0; sample < sampleNum; sample++)
                            {
                                if ((p.X < 0) || (p.X >= traceBox.Width) || clearAllTraces)
                                {    // here if initialising the trace of when the trace is
at the end of the box
                                    SolidBrush whiteBrush = new SolidBrush(Color.White);
                                    g.FillRectangle(whiteBrush, traceBox.ClientRectangle);
                                    whiteBrush.Dispose();
                                    tracePointLast[trace].X = 0;
                                    xLast[trace] = 0;
                                    p.X = 0;
                                }
                                p.Y = ((data[sample] + 4000) * traceBox.Height / 8000) - 1;
                                if (checkBox10sec.CheckState == CheckState.Checked)
                                {    // 10 sec means less than x axis 1 pixel for every
sample
                                    xLast[trace] += PixelsPerSample[trace];
                                    p.X = (int)xLast[trace];
                                }
                                else
                                    p.X++;
                                g.DrawLine(blackPen, tracePointLast[trace], p);
                                tracePointLast[trace] = p;
                            }
                            blackPen.Dispose();
                            g.Dispose();

                        }
```

If some samples have been returned by GetData(), the above code snippet draws them to the screen.

As always with C# drawing within a Windows application, a Graphics object is obtained for the drawing surface and a pen is used to draw connected lines.

If the waveform reaches the end of the trace box, the waveform is cleared and drawing restarts on the left. Note the boolean clearAllTraces that may have been set earlier to clear the traces.

Two x-axis drawing methods are used; if the trace is 10 seconds long, a floating point variable keeps track of the current x value otherwise, the x coordinate is simply incremented for each sample.

```
}
                }
                else if (gotAnalysis)
                {
                    gotAnalysis = false;
                    textBox1.Text += "Is the Analysis PC application installed and
working?\r\n";
                }
            }
            checkBox10secCheckStateLast = checkBox10sec.CheckState;
```

Now that drawing is complete, the above code snippet handles when ONLINE_OK is false or when an ONLINE_COMMSFAIL error occurs. A single error message is placed into the top textBox to warn the user.

```
if (sampleRateChanged)
            {
                string newRates = "Sampling rates for each trace = ";
                for (int trace = 0; trace < TRACENUM; trace++)
                    newRates += (sampleRate[trace].ToString() + ", ");

                textBox1.Text += (newRates + "\r\n");
            }
            // ******* End of this addition for Biometrics Example *******
        }
```

The sampling rates used by all 8 channels were collected earlier in the sampleRate array; a sampleRateChanged boolean flag was also set true if a change was detected. If changed (or at the start of the application), The rates are displayed in the top textBox. Note that any existing box text is added to rather than replaced.

```
private void buttonStartStop_Click(object sender, EventArgs e)
        {
            // ******* Start of an addition for Biometrics Example *******
            int status = 0;
            if (dataTransfer)
            {
                OnLineStatus(0, OnLineStatusType.ONLINE_STOP, ref status);
                buttonStartStop.Text = "Start Data Transfer";
                dataTransfer = false;
            }
            else
            {
                OnLineStatus(0, OnLineStatusType.ONLINE_START, ref status);
                if (status == (int)OnLineError.ONLINE_OK)
                {
                    buttonStartStop.Text = "Stop Data Transfer";
                    InitialiseAllTraces();
                    dataTransfer = true;
                }
            }
            // ******* End of this addition for Biometrics Example *******
        }
```

When the Start/Stop button is pressed, the above handler sends appropriate commands to the DLL and updates the text on the start/stop button. A check is made to ensure that a start actually took place before the button text is changed and the dataTransfer flag set.

```
private void buttonPrev8_Click(object sender, EventArgs e)
        {
            // ******* Start of an addition for Biometrics Example *******
            if (OnLineChannelOffset != 0)
                OnLineChannelOffset -= 9;            // skip 8 analogue channels and 1
digital channel
            int lastChannelOffset = OnLineChannelOffset + 7;
            textBoxUnitTitle.Text = "Data from channels " + OnLineChannelOffset.ToString()
+ " to " + lastChannelOffset.ToString();
            InitialiseAllTraces();
            // ******* End of this addition for Biometrics Example *******
        }

        private void buttonNext8_Click(object sender, EventArgs e)
        {
            // ******* Start of an addition for Biometrics Example *******
            if (OnLineChannelOffset != (7 * 9))    // maximum of 8 units having 9 channels
each
                OnLineChannelOffset += 9;        // skip 8 analogue channels and 1 digital
channel
            int lastChannelOffset = OnLineChannelOffset + 7;
            textBoxUnitTitle.Text = "Data from channels " + OnLineChannelOffset.ToString()
+ " to " + lastChannelOffset.ToString();
            InitialiseAllTraces();
            // ******* End of this addition for Biometrics Example *******
        }
        private void buttonClose_Click(object sender, EventArgs e)
        {
            // ******* Start of an addition for Biometrics Example *******
            Close();
            // ******* End of this addition for Biometrics Example *******
        }
    }
}
```

And finally, 2 button handlers allow the DataLITE channel for trace 1 to be changed and of course a button to close the application.

Visual Studio Comments

This project needs to add a reference to the Biometrics DLL. This can be done from the Project menu - Add Reference - COM Type Libraries - Biometrics DataLINK System Interface Library.

Because the 'unsafe' keyword is used, the application must be set to use unsafe code. In the Properties of the project, under the Build tab, set the enable use of unsafe code option.

8. MATLAB Interface

Overview

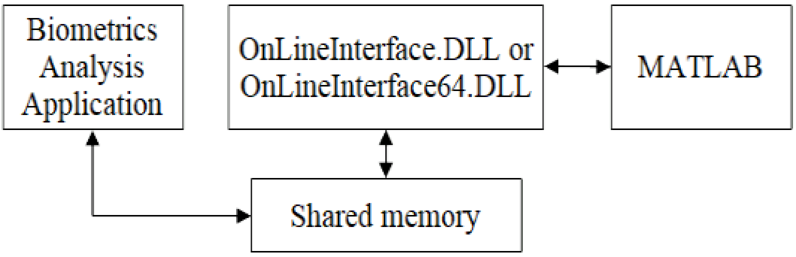
This is a simple introduction to using DataLITE, DataLOG or DataLINK with 32-bit and 64-bit MATLAB. The following MATLAB code has been tested using ***MATLAB R2015b*** (8.6.0) and ***R2017b*** (9.3.0).

The MATLAB code included is not optimised in any way and there may be better ways of performing the same actions within MATLAB. It is presented as operational code files that may be used as an introduction to the Biometrics interface.

The only difference between 32-bit and 64-bit MATLAB is the name of the DLL file; the DLL is ***OnLineInterface.DLL*** for 32-bit MATLAB or ***OnLineInterface64.DLL*** for 64-bit MATLAB. Note that *msvcr120_app.dll* is required (and supplied) for *OnLineInterface64.DLL*.

Before getting started...

The Biometrics Analysis application is responsible for gathering the data from the Biometrics hardware. From an external program's point of view, this application creates a shared block of memory into which the data is stored. When *OnLineInterface.DLL* (or *OnLineInterface64.DLL*) is loaded, it looks for this block of memory and provides two functions that can be used to interface with it and hence the Analysis application. By loading *OnLineInterface.DLL* (or *OnLineInterface64.DLL*), MATLAB is able to use 2 functions from the DLL to access data and control how the Biometrics Analysis program operates.


Biometrics Analysis Application
OnLineInterface.DLL or OnLineInterface64.DLL
MATLAB
Shared memory

There are some important points to remember.

- The Biometrics Application MUST be launched before MATLAB loads the DLL in order to create the shared memory.
- If the Biometrics Application is subsequently closed and reopened, it will see the existing shared memory and immediately close again.
- In order to collect data into shared memory and make it accessible to MATLAB, the Biometrics Application MUST have file save mode turned off (showing on the toolbar) and transfer started (by pressing the ✓ icon on the toolbar (or by a OLI.ONLINE_START status function call from MATLAB).
- Data is collected from the beginning of a start transfer into circular buffers that are a maximum of 50,000 samples in length for each channel. Data is removed by MATLAB from the beginning of the buffer and added by Analysis to the end. So long as MATLAB removes the data fast enough so that no more than 50,000 data values remain untaken from the buffer, no data is lost.
- Data is retained in the buffers after a stop transfer and this may be accessed by MATLAB. It is important that a buffer is not allowed to overflow because subsequent requests for data will return OLI.ONLINE_OVERRUN and no data.

Application loading

- Run the Biometrics application.
- Run MATLAB.
- Load the DLL by running the following lines (in DataLogInit.m supplied):

```
%       32-bit MATLAB
addpath('C:\Users\Public\Documents\MATLAB'); % ‘Public’ can be changed to your user profile
name on PC
if ~libisloaded('OnLineInterface')  % only load if not already loaded
    [notfound,warnings]=loadlibrary('OnLineInterface.dll', 'OnLineInterface.h');
end
%libfunctionsview('OnLineInterface')    % Open window showing functions in library
feature('COM_SafeArraySingleDim', 1);   % only use single dimension SafeArrays
feature('COM_PassSafeArrayByRef', 1);
```

```
%       64-bit MATLAB
addpath('C:\Users\Public\Documents\MATLAB'); % ‘Public’ can be changed to your user profile
name on PC
if ~libisloaded('OnLineInterface64')  % only load if not already loaded
    [notfound,warnings]=loadlibrary('OnLineInterface64.dll', 'OnLineInterface.h');
end
%libfunctionsview('OnLineInterface64')    % Open window showing functions in library
feature('COM_SafeArraySingleDim', 1);   % only use single dimension SafeArrays
feature('COM_PassSafeArrayByRef', 1);
```

  - The addpath function sets the directory containing the OnLineInterface.dll (or OnLineInterface64.DLL) and OnLineInterface.h files and this may be different to that shown.
  - The libisloaded function is used to prevent a reload of the DLL if it is already in memory.
  - The DLL OnLineInterface.dll (or OnLineInterface64.DLL) is loaded along with a C header file supplied (OnLineInterface.h) that defines a couple of structures used by the interface and functions contained in the library DLL.
  - The libfunctionsview function is optional and for information only.
  - The two feature lines tell MATLAB that the interface uses single dimension arrays to hold the data.
  - As with all code blocks in this document, the above would normally be integrated into a more elegant final MATLAB application!

The *OnLineInterface.h* file is listed below for information:

```
%       32-bit and 64-bit MATLAB
typedef struct tagSAFEARRAYBOUND {
    unsigned long cElements;
    long lLbound;
} SAFEARRAYBOUND;

typedef struct tagSAFEARRAY {
    unsigned short cDims;       // Count of dimensions in this array.
    unsigned short fFeatures;   // Flags used by the SafeArray
    unsigned long cbElements;   // Size of an element of the array.
    unsigned long cLocks;       // Number of times array has locked without unlock.
    void * pvData;              // Pointer to the data.
    SAFEARRAYBOUND rgsabound;   // One bound for each dimension.
} SAFEARRAY;

__declspec(dllexport) long OnLineGetData(
                             int ch, int sizeMs, SAFEARRAY **pData, int *pActualSamples);
__declspec(dllexport) long OnLineStatus(


                       int ch, int statusType, int *pStatus);
```

A useful helper class called OLI (easier than typing OnLineInterface!) is supplied that simply defines the constants used by the OnLineInterface functions; it is listed below for information:

```
classdef OLI
    properties( Constant = true )
    ONLINE_GETERROR     = 0
    ONLINE_GETENABLE    = 1 % 1 if specified channel is enabled, 0 if not
    ONLINE_GETRATE      = 2 % number of samples per second on the specified channel
    ONLINE_GETSAMPLES   = 3 % the number of unread samples on the specified channel
    ONLINE_GETVALUE     = 4 % the (current value + 4000) on the specified channel
    ONLINE_START        = 5 % start or re-start data transfer
    ONLINE_STOP         = 6 % stop data transfer

    ONLINE_OK           = 0
    ONLINE_COMMSFAIL    = -3
    ONLINE_OVERRUN      = -4
    end
end
```

OnLineStatus Function

```
%       32-bit MATLAB
calllib('OnLineInterface', 'OnLineStatus', channel, statusType, pStatusValue);
```

```
%       64-bit MATLAB
calllib('OnLineInterface64', 'OnLineStatus', channel, statusType, pStatusValue);
```

The code below (supplied in getSampleRate.m) illustrates how to get the current sampling rate for channel *ch* using the function OnLineStatus from the library DLL:

```
%       32-bit MATLAB
pStatus = libpointer('int32Ptr', 0);
calllib('OnLineInterface', 'OnLineStatus', ch, OLI.ONLINE_GETRATE, pStatus);
get(pStatus, 'Value');  % and display the information
```

```
%       64-bit MATLAB
pStatus = libpointer('int32Ptr', 0);
calllib('OnLineInterface64', 'OnLineStatus', ch, OLI.ONLINE_GETRATE, pStatus);
get(pStatus, 'Value');  % and display the information
```

- libpointer initialises a variable of the correct type for use with the DLL function OnLineStatus.
- The calllib function goes to the defined DLL and executes the defined function.
- OLI.ONLINE_GETRATE is a constant of value 2 defined in the class of constants called OLI; this instructs the OnLineStatus function to returns the sampling rate of channel 0 in the variable pStatus.
- get is a simple way to query the properties of pstatus.
- The only difference between 32-bit and 64-bit MATLAB is the name of the DLL.

The next to the last parameter must be one of the OLI class constants listed below. Note that any command will return OLI.ONLINE_COMMSFAIL in pStatus if communications with the hardware fails.

Status Constant	**Value**	**Comment**
OLI.ONLINE_GETERROR	0	channel is unused. Return with the current status in pStatus.
OLI.ONLINE_GETENABLE	1	pStatus returns with 1 if the specified channel is enabled or 0 if it is not.
OLI.ONLINE_GETRATE	2	pStatus returns with the number of samples per second on the specified channel.
OLI.ONLINE_GETSAMPLES	3	pStatus returns with the number of unread samples on the specified channel or OLI.ONLINE_OVERRUN.

OLI.ONLINE_GETVALUE	4	pStatus returns with the (current value + 4000) on the specified channel (i.e. between 0 and 8000)
OLI.ONLINE_START	5	channel is unused. Start or re-start the data transfer.
ONLINE_STOP	6	channel is unused. Stop the data transfer.

The contents of pStatus on return contains the desired information or one of the following:

Error Return Constant	**Value**	**Comment**
OLI.ONLINE_OK	0	No communications or buffer errors.
OLI.ONLINE_COMMSFAIL	-3	Communications with the hardware has failed.
OLI.ONLINE_OVERRUN	-4	The internal buffer has overflowed and some data has been lost.

Note that samples returned by this function using OLI.ONLINE_GETSAMPLES will only change at a very slow rate of around one per second. To increase this rate, use OLI.ONLINE_START (or press the ✓ icon on the Biometrics toolbar) to begin data transfer before using OLI.ONLINE_GETSAMPLES repeatedly.

If more than one DataLOG or DataLINK unit is being used simultaneously to collect data then the MATLAB channel number needs to reflect which unit is being accessed. Each unit has 9 channels of data (8 analogue and one 4-bit digital) which are accessed in sequence. The MATLAB channel number is therefore:

(unit number * 9) + channel number within the unit - 1

where the unit number can be 0 to 7 and the DataLOG channel number can be 1 to 9. The unit number is its order in the Detected Units Window with unit 0 being at the top. Note that units marked without Transfer or with Removed are still counted.

If a recording is made using DataLITE, The MATLAB channel number is:

((Record Window - 1) * 9) + channel number within the Record Window - 1

where the DataLITE Record Window can be 1, 2 or 3 and the DataLITE channel number can 1 to 9; note that the DataLITE digital channel (8, 17 or 26) contains 4-bits which is a combination of 2 digital DataLITE probe pairs.

For example, a Biometrics pinchmeter sensor on DataLITE channel 1 of recording window 3 is MATLAB channel 18.

See the supplied function in getSamplesAvailable.m for another example.

OnLineGetData Function

```
%       32-bit MATLAB
calllib('OnLineInterface', 'OnLineGetData', channel, sizeMsToRead, DataArray,
pActualSamples);
```

```
%       64-bit MATLAB
calllib('OnLineInterface64', 'OnLineGetData', channel, sizeMsToRead, DataArray,
pActualSamples);
```

This function return the next block of data received on the specified *channel*. Analogue channels are returned when *channel* is 0 to 7 (or 9 to 25, 27 to 34, ...); *channel* 8 (or 17, 26, ..) returns the digital values as a set of 4-bits.

In order to collect data and make it accessible to MATLAB, the Biometrics Application MUST have file save mode turned off (showing on the toolbar) and transfer started (by pressing the ✓ icon on the toolbar or by an OLI.ONLINE_START status function call from MATLAB).

The number of samples returned in the *DataArray* array will be no more than (*sizeMsToRead* * sample rate) where *sizeMsToRead* is a number of milliseconds. If insufficient samples are available then all that is available will be returned.

If no error occurred, *pActualSamples* will return with the actual number of samples returned in the *DataArray* array. If an error did occur, *pActualSamples* will return with a negative value (possibly OLI.ONLINE_COMMSFAIL or OLI.ONLINE_OVERRUN).

The following MATLAB function, getData (supplied in getData.m), illustrates using the OnLineGetData function.

```
%       32-bit (and 64-bit MATLAB but see below)
function numberOfValues = getData(ch, sampleRate, numberOfValues, values)
% Get the next block of data from Biometrics DataLog.
%   ch is the required channel.
%   sampleRate is the number of samples per second on the required channel.
%   numberOfValues is the maximum number of samples to get.
%   The data is returned in an array called 'values.pvData'.
%   The function returns the number of samples in this array.

% First initialise the return values from the DLL functions
pStatus = libpointer('int32Ptr', 0);
% Find how many samples are available
calllib('OnLineInterface', 'OnLineStatus', ch, OLI.ONLINE_GETSAMPLES, pStatus);

if (pStatus.Value < 0)
    if (pStatus.Value == OLI.ONLINE_OVERRUN)
        error('The Biometrics Analysis buffer has overrun! More than 50000 samples not
transferred by MATLAB so stop and re-start the recording.');
    else
        error('Is the DataLog switched on?');
    end;
end;

if (numberOfValues > pStatus.Value)
    numberOfValues = pStatus.Value; % Never try to get more values than are available
end;

% initialise the return data array
Data = int16(1:numberOfValues);
% initialise the return value from OnLineGetData within DLL
pDataNum = libpointer('int32Ptr', 0);
values.cDims = int16(1);
values.cbElements = 2;  % 2-byte values
values.cLocks = 0;
values.rgsabound.cElements = numberOfValues;
values.rgsabound.lLbound = numberOfValues;
values.pvData = Data;
```

```
calllib('OnLineInterface', 'OnLineGetData', ch, numberOfValues * 1000 / sampleRate, values,
pDataNum);
numberOfValues = pDataNum.Value;
```

```
%       64-bit MATLAB
As other 64-bit MATLAB examples, simply change the first parameter of callib from
OnLineInterface to OnLineInterface64 in the code above.
```

- *OLI.ONLINE_GETSAMPLES* is a constant of value 3 defined in the class of constants called OLI; this instructs the OnLineStatus function to returns the number of unread samples on the channel ch in the variable pStatus.
- If the OnLineStatus function returns a negative value in *pStatus*, then an error has occurred (possibly OLI.ONLINE_COMMSFAIL or OLI.ONLINE_OVERRUN).
- *Data = int16(1:numberOfValues)* sets up an array containing 2-byte values of length *numberOfValues*.

The MATLAB getData function may be used as follows (supplied in DataLogGraph.m):

```
%       32-bit and 64-bit MATLAB
int32 ch;
ch = 18;  % will need to be changed to match the sensor channel
values = libstruct('tagSAFEARRAY');
numberOfValues = getData(ch, 2000, 4000, values);
if (numberOfValues > 0)
    plot(values.pvData);
else
    str = ['getData() returned ',num2str(numberOfValues),' from channel',num2str(ch)];
    disp(str);
end
```

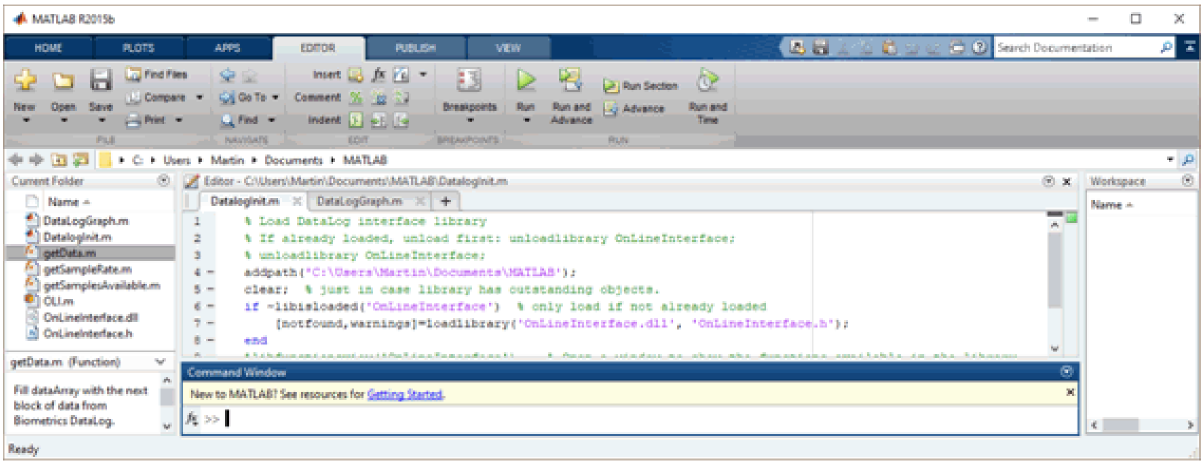
- *values* is a structure of type *tagSAFEARRAY* as defined in the OnLineInterface.h file; *getData* will copy the data into the *pvData* part of this.
- The *getData* function requires a channel number (ch above), the channel sampling rate (2000 above) and the number of values (4000 above).
- *plot* is a built in MATLAB function that can be used to display the data.

Pulling it together

If MATLAB is new to you, try the following in the order suggested:

Click here to download project files.

- Copy the following files to a working directory on your PC. (C:\Users\Public\Documents\MATLAB in this example):
    - DatalogInit.m
    - DataLogGraph.m
    - FunctionTest.m
    - getData.m
    - getSampleRate.m
    - getSamplesAvailable.m
    - OLI.m
    - OnLineInterface.dll (or OnLineInterface64.DLL)
    - OnLineInterface.h
    - msvcr120_app.dll if 64-bit MATLAB
- Launch the Biometrics Application.
- The Biometrics Application MUST have file save mode turned off (☒ showing on the toolbar).
- Open MATLAB and just below the menu bar, browse for the user working directory to show the files just copied into the Current Folder window:

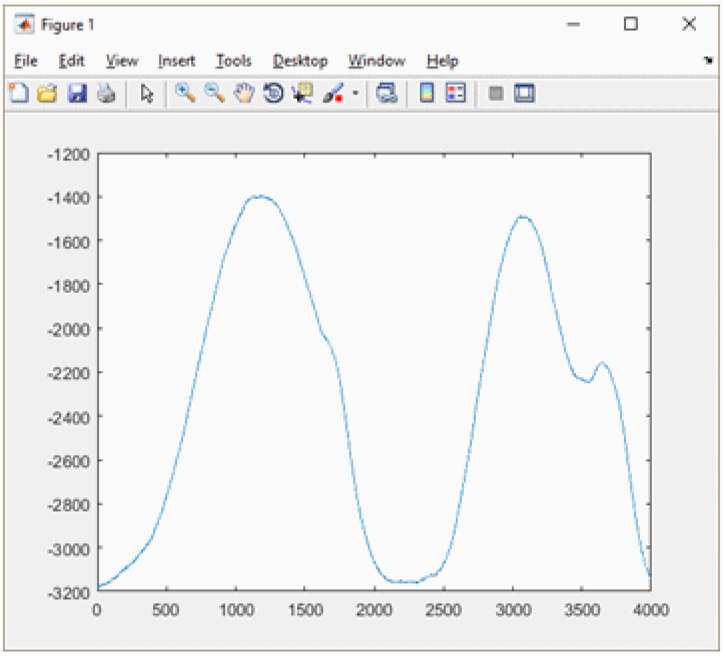


```
% Load DataLog interface library
% If already loaded, unload first: unloadlibrary OnLineInterface;
% unloadlibrary OnLineInterface;
addpath('C:\Users\Martin\Documents\MATLAB');
clear;  % just in case library has outstanding objects.
if ~libisloaded('OnLineInterface')  % only load if not already loaded
    [notfound,warnings]=loadlibrary('OnLineInterface.dll', 'OnLineInterface.h');
end
```

New to MATLAB? See resources for Getting Started.

```
>>
```

- Open *DatalogInit.m* in the editor and click Run.
- Setup a sensor and record it in the Biometrics Application (by pressing the ✓ icon on the toolbar).
    - For a more leisurely test, try collecting a little data before running *DataLogGraph.m*.
    - Data is collected in a 50,000 sample buffer which must not be allowed to overflow. For the sake of this example, try starting a recording and then stopping it before the buffer fills - a recording of less than 50000/sps seconds for example (i.e. less than 25 seconds at 2000sps).
- Open *DatalogGraph.m* in the editor, change the value of *ch* to match a channel that is used and click Run to show a graph containing the data collected:



If MATLAB says that it does not understand *tagSAFEARRAY*, try closing MATLAB and restart MATLAB again; run *DatalogInit.m* or similar again to reload the library and the contents of *OnLineInterface.h*.

- Re-run *DatalogGraph.m* to update the graph with the next block of up to 4000 samples.
- Open *FunctionTest.m* in the editor, change the value of ch to match a channel that is used and click Run.

The *FunctionTest.m* file is listed below for information:

```
%       32-bit MATLAB
int32 ch;
ch = 18;    % Change to match a channel that is used
sampleRate = getSampleRate(ch);
str = ['Channel ', num2str(ch), ' has a sampling rate of ', num2str(sampleRate)];
disp(str);

samplesAvailable = getSamplesAvailable(ch);
str = ['Channel ', num2str(ch), ' has ', num2str(samplesAvailable), ' samples available.'];
disp(str);

pStatus = libpointer('int32Ptr', 0);
calllib('OnLineInterface', 'OnLineStatus', ch, OLI.ONLINE_GETENABLE, pStatus);
enabled = pStatus.Value;
```

```
%       64-bit MATLAB
int32 ch;
ch = 18;    % Change to match a channel that is used
sampleRate = getSampleRate(ch);
str = ['Channel ', num2str(ch), ' has a sampling rate of ', num2str(sampleRate)];
disp(str);

samplesAvailable = getSamplesAvailable(ch);
str = ['Channel ', num2str(ch), ' has ', num2str(samplesAvailable), ' samples available.'];
disp(str);

pStatus = libpointer('int32Ptr', 0);
calllib('OnLineInterface64', 'OnLineStatus', ch, OLI.ONLINE_GETENABLE, pStatus);
enabled = pStatus.Value;
```