Decision Industrial Controller
Decision Industrial Controller Client OCX Component

Users Manual

1998/9/20 Rick Wang in CYCU

- **Functions to connect the remote report server**

DicOcx.BuildConnect

This function ask user to input the connect information , such like link port and address. Before you use the DIC Ocx component, you must connect to the **report server** first.

Declaration

BOOL BuiltConnect

Return Value

TRUE if connection is successful, FALSE otherwise.

- **Functions to enumerate or browse devices at the remote server**

DicOcx.SelectDevice

This function displays a dialog box on the screen and allows the user to select one of the installed devices. The name of the device is returned, and you can select the exist device as default work device.

Declaration

BOOL SelectDevice

Return value

TRUE if successful and the user has confirmed his/her selection, FALSE otherwise.

- **Functions to for digital input/output**

DicOcx.SetDigitalBit

This function sets or clears a single bit on a digital output line.

Declaration

```
BOOL SetDigitalBit  (   DWORD dwLine,
                        BOOL    bState
                    );
```

Parameters

dwLine The index of the bit on the card to manipulate. The first bit has index 0.

bState The new state of the bit, either set (1/TRUE) or cleared (0/FALSE)

Return value

TRUE if successful, FALSE otherwise.

DicOcx.SetDigitalByte

This function outputs a complete byte to a digital output port of a device.

Declaration

```
BOOL SetDigitalByte    (       DWORD dwPort,
                               BYTE    byPortState
                       );
```

Parameters

dwPort The index of the port on the card to manipulate. The first port has index 0.

byPortState The new state of the port

Return value

TRUE if successful, FALSE otherwise.

DicOcx.GetDigitalBit

This function returns the state of a single bit on an input port of a device

Declaration

```
BOOL GetDigitalBit    (    DWORD dwLine,
                           LPBOOL lpbState
                      );
```

Parameters

dwLine The index of the bit on the card to manipulate. The first bit has index 0.

lpbState A pointer to a variable receiving the new state of the bit, either set (1/TRUE) or cleared (0/FALSE)

Return value

TRUE if successful, FALSE otherwise.

DicOcx.GetDigitalByte

This function input a complete byte from a digital input port of a device.

Declaration

```
BOOL GetDigitalByte    (        DWORD dwPort,
                                LPBYTE lpbyPortState
                       );
```

Parameters

dwPort The index of the port on the card to read. The first port has index 0.

lpbyPortState A pointer to a variable of type BYTE receiving the new state of the port

Return value

TRUE if successful, FALSE otherwise.

DicOcx.GetOutputPort

This function reads the state of an output port, i.e. a bank of relays.

Declaration

```
BOOL GetOutputPort   (      DWORD dwPort,
                            LPBYTE lpbyPortState
                     );
```

Parameters

dwPort The index of the *output* port on the card to read. The first port has index 0.

lpbyPortState A pointer to a variable of type BYTE receiving the new state of the port

Return value

TRUE if successful, FALSE otherwise.

DicOcx.Set8255Config

This function configures a 8255 chip on the device.

Declaration

```
BOOL Set8255Config   (      DWORD dwChip,
                            BYTE byConfiguration
                     );
```

Parameters

dwChip The index of the chip on the card to manipulate. The first chip has index 0.

byConfiguration A byte containing the new configuration for the 8255 chip. This byte must conform to the configuration byte specified in the 8255 chip's data sheet.

Return value

TRUE if successful, FALSE otherwise.

- **Functions to for analog input / output**

DicOcx.SetAnalogChannel

This function sets an analog channel on the device to a specific value. It takes a 32-bit integer value as a RAW value to be written to the channel.

Declaration

```
BOOL SetAnalogChannel   (   DWORD dwChannel,
                            DWORD dwValue
                        );
```

Parameters

dwChannel The index of the channel on the card to manipulate. The first channel has index 0.

dwValue A 32-bit integer containing the value to output. The value is truncated according to the appropriate resolution of the device

Return value

TRUE if successful, FALSE otherwise.

DicOcx.SetRealAnalogChannel

This function sets an analog channel on the device to a specific value. It takes a double value in the range supported by the device and/or setup by the user.

Declaration

```
BOOL SetRealAnalogChannel  (     DWORD dwChannel,
                                 double dValue
                           );
```

Parameters

dwChannel The index of the channel on the card to manipulate. The first channel has index 0.

dValue A double (floating point) value containing the value to be set. It is converted to a raw device value depending on the range setup for the device and the resolution supported by the device. This conversion is performed automatically by the DIC.

Return value

TRUE if successful, FALSE otherwise.

DicOcx.GetAnalogChannel

This function sets an analog channel on the device to a specific value. It returns the raw integral value from the card.

Declaration

```
BOOL GetAnalogChannel   (   DWORD dwChannel,
                            LPDWORD lpdwValue
                        );
```

Parameters

dwChannel The index of the channel on the card to read. The first channel has index 0.

lpdwValue A pointer to a 32-bit integer receiving the current value of the analog channel. The value is truncated according to the appropriate resolution of the device

Return value

TRUE if successful, FALSE otherwise.

DicOcx.GetRealAnalogChannel

This function sets an analog channel on the device to a specific value. It returns an interpreted value from the card.

Declaration

```
BOOL GetRealAnalogChannel  (      DWORD dwChannel,
                                  double * lpdValue
                           );
```

Parameters

dwChannel The index of the channel on the card to read. The first channel has index 0.

lpdValue A pointer to a double (floating point) variable receiving the current value of the analog channel.
This value is interpreted according to the channel value range setup by the user and supported by the card.

Return value

TRUE if successful, FALSE otherwise.

- **Functions to configure analog input / output channels**

Furthermore, the DIC can also automatically convert all values to a specified range setup by the user. If, for example, a device is used for measuring currents in the range from 0-20 mA, equating 0-10 V, the DIC can directly return the range 0-20 mA, by setting it up in the function DicOcx.SetChannelRange.

Please note that the function DicOcx.SetChannelRange is merely performing a calculation inside the DIC, but does not have any effect on the underlying hardware.

Functions that change the configuration of the hardware, with respect to the supported range of values, are DicOcx.SetChannelGain and DicOcx.SetChannelBipolar.

DicOcx.SetChannelGain

This function configured a programmable amplifier/gain on an analog device.

Declaration

```
BOOL  SetChannelGain  (  BOOL bInput,
                         DWORD dwChannel,
                         double dGain
                      );
```

Parameters

bInput Specified whether the gain is to be set on an input channel (TRUE) or output channel (FALSE)

dwChannel The index of the channel on the card. The first channel has index 0.

dGain A floating point value indicating the gain to be setup for the channel.

Return value

TRUE if successful, FALSE otherwise.

DicOcx.SetChannelBipolar

This function configured a programmable bipolar mode on an analog device.

Declaration

```
BOOL SetChannelBipolar    (    BOOL bInput,
                               DWORD dwChannel,
                               BOOL bIsBipolar
                          );
```

Parameters

bInput Specified whether the channel is an input channel (TRUE) or output channel (FALSE)

dwChannel The index of the channel on the card. The first channel has index 0.

bIsBipolar A boolean indicating that the channel is bipolar (TRUE) or unipolar (FALSE).

Return value

TRUE if successful, FALSE otherwise.

DicOcx.SetChannelRange

This function configured a user-defined channel range for the analog device. All values retrieved from DicOcx.GetRealAnalogChannel or setup by DicOcx.SetRealAnalogChannel will be inside this range.

Declaration

```
BOOL SetChannelRange    (    BOOL bInput,
                             DWORD dwChannel,
                             double dMinimum
                             double dMaximum
                        );
```

Parameters

bInput Specified whether the channel is an input channel (TRUE) or output channel (FALSE)

dwChannel The index of the channel on the card. The first channel has index 0.

dMinimum Specifies the minimum value of the channel

dMaximum Specifies the maximum value of the channel

Return value

TRUE if successful, FALSE otherwise.

DicOcx.GetChannelRange

This function returns a user-defined channel range for the analog device. All values retrieved from DicOcx.GetRealAnalogChannel or setup by DicOcx.SetRealAnalogChannel will be inside this range.

Declaration

```
BOOL GetChannelRange    (    BOOL bInput,
                             DWORD dwChannel,
                             double * lpdMinimum
                             double * lpdMaximum
                        );
```

Parameters

bInput Specified whether the channel is an input channel (TRUE) or output channel (FALSE)

dwChannel The index of the channel on the card. The first channel has index 0.

dMinimum A variable receiving the minimum value of the channel

dMaximum A variable receiving the maximum value of the channel

Return value

TRUE if successful, FALSE otherwise.

DicOcx.GetDeviceChannelRange

This function returns the device-dependent value range. It takes into account the physical hardware limitations and the current configuration of amplifiers.

Declaration

```
BOOL GetDeviceChannelRange    (    BOOL bInput,
                                   DWORD dwChannel,
                                   double * lpdMinimum
                                   double * lpdMaximum
                               );
```

Parameters

bInput Specified whether the channel is an input channel (TRUE) or output channel (FALSE)

dwChannel The index of the channel on the card. The first channel has index 0.

dMinimum A variable receiving the minimum value of the channel

dMaximum A variable receiving the maximum value of the channel

Return value

TRUE if successful, FALSE otherwise.

- **Functions to access timers on cards**

DicOcx.SetTimerConfig

This function configures timer chips on the device.

Declaration

```
BOOL SetTimerConfig  (      DWORD dwTimer,
                            DWORD dwConfig
                     );
```

Parameters

dwTimer The index of the timer on the card to access. The first timer has index 0.

dwConfig A 32-bit integer containing the configuration for the timer. On 8253 timers, only the lowest byte is being used.

Return value

TRUE if successful, FALSE otherwise.

DicOcx.LoadTimer

This function loads a value into a timer.

Declaration

```
BOOL LoadTimer    (    DWORD dwTimer,
                       DWORD dwValue
                  );
```

Parameters

dwTimer The index of the timer on the card to access. The first timer has index 0.

dwValue A 32-bit integer containing the timer value to write.

Return value

TRUE if successful, FALSE otherwise.

DicOcx.GetTimer

This function loads a value into a timer.

Declaration

```
BOOL GetTimer    (    DWORD dwTimer,
                      LPDWORD lpdwValue
                  );
```

Parameters

dwTimer The index of the timer on the card to access. The first timer has index 0.

lpdwValue A pointer to a 32-bit integer receiving the current timer value.

Return value

TRUE if successful, FALSE otherwise.

- **Functions to handle standard notification**

The following structures and constants are used for the notification mechanism. Please note that these may be specific to the C language. Other language may use the OCX for easier access to notification functions. In the OCX, notifications are implemented as standard OCX events for simplified use.

Note: Notifications as well as OCX Events are only supported for future I/O devices that support hardware interrupts. We plan on providing notification support for cards (about remote server), as well, at a later time. We recommend that you check the boolean return value of **DicOcx.RequestNotification** to make sure the device supports notifications.

DicOcx.RequestNotification

This function requests notifications from the device for specific events that may happen.

Declaration

```
BOOL RequestNotification    (    DWORD dwNotificationMask,
                                 DWORD dwUserValue
                            );
```

Parameters

dwNotificationMask
A 32-bit integer that specifies the notifications that are requested from the device

dwUserValue A 32-bit integer that gets passed back to the user inside the DII_NOTIFICATION_DATA structure

Return value

TRUE if successful, FALSE otherwise.

DicOcx.CancelNotification

This function cancels previously requested notifications again. No more notifications for the events canceled will be sent to the user application.

Declaration

```
BOOL CancelNotification     (     DWORD dwNotificationMask
                           );
```

Parameters

dwNotificationMask

A 32-bit integer that specifies the notifications that should be canceled from the device (Currently ignored)

Return value

TRUE if successful, FALSE otherwise.

● Functions to access kernel mode pulse counters

The Pulse Counter Interface was created to satisfy the demand of detecting digital pulses from high-level languages. Previously, these pulses could only be detected by either polling the hardware I/O ports directly from a high-level language, or, if the device supports interrupts, processing these interrupts within the high-level languages.
Both approaches have disadvantages, the polling is not reliably on multi-tasking operating systems such as Windows NT and Windows 95, and processing interrupts in a high-level language also causes a lot of overhead.

The Pulse Counting Interface offers the following functionality:

1. Every channel on an digital I/O card can individually be configured as an up or down counter, can individually be reset and preset at any given time.
2. Along with the actual number of pulses, the minimum and maximum time between two pulses is also returned. Note that this information is subject to the resolution of your hardware platform, which may vary.
3. Down-counters can individually be configured to automatically preset themselves when the counter has reached zero (thus to keep counting), or to stay at zero until manually reset again.

The Pulse Counting interface is implemented with the following functions:

DicOcx.EnablePulseCounting

This function enables pulse counting for the card specified.

Declaration

```
BOOL EnablePulseCounting    (       DWORD dwStartPort,
                                    DWORD dwNumberOfPorts,
                                    DWORD dwReserved
                            );
```

Parameters

dwStartPort The start I/O port, which shall be monitored. To monitor all ports, specify 0.

dwNumberOfPorts
The number of ports to support or scan. To monitor all ports, specify 0.

dwReserved Reserved for future use, specify 0.

Return value

TRUE if successful, FALSE otherwise.

DicOcx.DisablePulseCounting

This function disables pulse counting for the card specified.

Declaration

```
BOOL DisablePulseCounting    ( void
                               );
```

Return value

TRUE if successful, FALSE otherwise.

DicOcx.SetDownPulseCounter

This function is called to reconfigure a counter into up-or down pulse counter

Declaration

```
BOOL SetDownPulseCounter   (     DWORD dwChannel,
                                 DWORD dwInitialValue,
                                 BOOL bAutoReset
                           );
```

Parameters

dwChannel The channel to reconfigure. The channel corresponds to a single digital I/O line. The first I/O line is specified with 0.

dwInitialValue The initial value for the counter. If non-zero, the counter is configured to be a down-counter, if zero is specified, the counter is configured as an up-counter. The counter is initialized to the this value.

bAutoReset Specifies whether an down-counter shall automatically reset itself (to the initial value specified), when it counts down to zero. When the counter is configured as an up-counter (dwInitialValue is zero), this parameter is ignored.

Return value

TRUE if successful, FALSE otherwise.

DicOcx.GetPulseCounterValue

This function is called to reconfigure a counter into up-or down pulse counter

Declaration

```
BOOL GetPulseCounterValue   (       DWORD dwChannel,
                                    LPDWORD lpdwValue,
                                    LPDWORD lpdwMinTimeBetweenPulses,
                                    LPDWORD lpdwMaxTimeBetweenPulses,
                                    BOOL bResetCounter
                            );
```

Parameters

dwChannel The channel to reconfigure. The channel corresponds to a single digital I/O line. The first I/O line is specified with 0.

lpdwValue A pointer to a variable receiving the current contents of the counter.

lpdwMinTimeBetweenPulses
A pointer to a variable receiving the minimum time between two pulses. The time is given in *microseconds*, although the actual timer resolution of computer will be lower.

lpdwMinTimeBetweenPulses
A pointer to a variable receiving the maximum time between two pulses. The time is given in *microseconds*, although the actual timer resolution of computer will be lower.

bResetCounter Specifies whether the counter shall be reset while reading. If reset, an up-counter is set to zero, and a down-counter is set to the inital value again. If this parameter is FALSE, the counter is not reset.

Return value

TRUE if successful, FALSE otherwise.

- **Properties**

The following properties are exposed by the OCX control:

Property

State

Type

Integer

Persistent

Remark

To receive the connect status right now. The state condition as follow ..

0 **Default, Close**
1 **Open**
2 **Listen**
3 **Pause**
4 **Check for Server**
5 **Check for Server at all**
6 **Connecting**
7 **has Connected**
8 **Closing now**
9 **Error**

Property

DeviceName

Type

String

Persistent

Example (Visual Basic)

```
'' The Example assumes you have a form containing the DIC-OCX control named "DiiDevice"
'' The device name is listed form report server (DIC Server)
'' See the following Chapter for information about how to use the OCX from Visual Basic

DicOcx.DeviceName = "Device0"

If Not DicOcx.DeviceOpened Then

        MsgBox "Unable To Open Device named Device0"

End If
```

Remarks

This property contains the device name to be opened by an instance of the OCX. As soon as this property is being changed, the OCX closes the existing devices and attempts to reopen the device with the new name specified.

You may also use the OCX method "SelectDevice" to display a dialog box in which the user can select a device name to operate upon.

In order to determine if the device name actually matched a device installed in your machine and is ready for access, you may use the property "DeviceOpened", which is TRUE, if the device was opened successfully, or FALSE otherwise.

Property

Exclusive

Type

Boolean

Persistent

Example (Visual Basic)

```
'' The Example assumes you have a form containing the DIC-OCX control named "DiiDevice".
'' See the following Chapter for information about how to use the OCX from Visual Basic

DicOcx.DeviceName = "Device0"

DicOcx.Exclusive = True

If Not DicOcx.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

End If
```

Remarks

This property determines how to open an adapter, either exclusively or non-exclusively. Setting this property to TRUE will prevent other applications or other OCXs from opening the device with the same name.

Property

DeviceOpened

Type

Boolean

Read-Only

Example (Visual Basic)

```
'' The Example assumes you have a form containing the DIC-OCX control named "DiiDevice".
'' See the following Chapter for information about how to use the OCX from Visual Basic

DicOcx.DeviceName = "Device0"

If Not DicOcx.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

End If
```

Remarks

This property is set to TRUE, whenever the OCX has successfully opened a device, FALSE otherwise.

Property

DigitalChannels

Type

long

Read-Only

Example (Visual Basic)

```
‘’ The Example assumes you have a form containing the DIC-OCX control named “DiiDevice”.
‘’ See the following Chapter for information about how to use the OCX from Visual Basic

DicOcx.DeviceName = “Device0”

If Not DicOcx.DeviceOpened Then

    MsgBox “Unable To Open Device named Device0”

Else

    If DicOcx.DigitalChannels = 0 Then

        MsgBox “This device cannot do digital I/O”

    End If

End If
```

Remarks

This property contains the number of digital channels for the currently opened Industrial I/O device, as specified in the property "DeviceName".

Note: You can also use the *DigitalInputChannels* and *DigitalOutputChannels* properties to differentiate between input and output channels.

Property

DigitalInputChannels

Type

long

Read-Only

Example (Visual Basic)

```
'' The Example assumes you have a form containing the DIC-OCX control named "DiiDevice".
'' See the following Chapter for information about how to use the OCX from Visual Basic

DicOcx.DeviceName = "Device0"

If Not DicOcx.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

Else

    If DicOcx.DigitalInputChannels = 0 Then

        MsgBox "This device does not have digital input lines"

    End If

End If
```

Remarks

This property contains the number of digital input channels for the currently opened Industrial I/O device, as specified in the property "DeviceName".

Property

DigitalOutputChannels

Type

long

Read-Only

Example (Visual Basic)

```
'' The Example assumes you have a form containing the DIC-OCX control named "DiiDevice".
'' See the following Chapter for information about how to use the OCX from Visual Basic

DicOcx.DeviceName = "Device0"

If Not DicOcx.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

Else

    If DicOcx.DigitalOutputChannels = 0 Then

        MsgBox "This device does not have digital output lines"

    End If

End If
```

Remarks

This property contains the number of digital output channels for the currently opened Industrial I/O device, as specified in the property "DeviceName".

Property

AnalogChannels

Type

long

Read-Only

Example (Visual Basic)

```
'' The Example assumes you have a form containing the DIC-OCX control named "DiiDevice".
'' See the following Chapter for information about how to use the OCX from Visual Basic

DicOcx.DeviceName = "Device0"

If Not DicOcx.DeviceOpened Then

        MsgBox "Unable To Open Device named Device0"

Else

        If DicOcx.AnalogChannels = 0 Then

                MsgBox "This device cannot do analog I/O"

        End If

End If
```

Remarks

This property contains the number of analog channels for the currently opened Industrial I/O device, as specified in the property "DeviceName".

Note: You can also use the *AnalogInputChannels* and *AnalogOutputChannels* properties to differentiate between input and output channels.

Property

AnalogInputChannels

Type

long

Read-Only

Example (Visual Basic)

```
'' The Example assumes you have a form containing the DIC-OCX control named "DiiDevice".
'' See the following Chapter for information about how to use the OCX from Visual Basic

DicOcx.DeviceName = "Device0"

If Not DicOcx.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

Else

    If DicOcx.AnalogInputChannels = 0 Then

        MsgBox "This device does not have analog input channels"

    End If

End If
```

Remarks

This property contains the number of analog input channels for the currently opened Industrial I/O device, as specified in the property "DeviceName".

Property

AnalogOutputChannels

Type

long

Read-Only

Example (Visual Basic)

```
'' The Example assumes you have a form containing the DIC-OCX control named "DiiDevice".
'' See the following Chapter for information about how to use the OCX from Visual Basic

DicOcx.DeviceName = "Device0"

If Not DicOcx.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

Else

    If DicOcx.AnalogOutputChannels = 0 Then

        MsgBox "This device does not have analog output channels"

    End If

End If
```

Remarks

This property contains the number of analog output channels for the currently opened Industrial I/O device, as specified in the property "DeviceName".

Property

Resolution

Type

long

Read-Only

Example (Visual Basic)

```
'' The Example assumes you have a form containing the DIC-OCX control named "DiiDevice".
'' See the following Chapter for information about how to use the OCX from Visual Basic

DicOcx.DeviceName = "Device0"

If Not DicOcx.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

Else

    MaximumResolution = DicOcx.Resolution

End If
```

Remarks

This property contains the resolution in bits for the currently opened Industrial I/O device, as specified in the property "DeviceName".

Property

NotificationMask

Type

long

Example (Visual Basic)

```
'' The Example assumes you have a form containing the DIC-OCX control named "DiiDevice".
'' See the following Chapter for information about how to use the OCX from Visual Basic

DicOcx.DeviceName = "Device0"

If Not DicOcx.DeviceOpened Then

        MsgBox "Unable To Open Device named Device0"

Else

        DicOcx.NotificationMask = &H2
                        '' Enables Signal Change Notifications

                        '' Check, if notifications were activated:
        If Not DicOcx.NotificationsActive Then
                MsgBox "This device does not support notifications"
        End If

                        '' Disable notifications again.
        DicOcx.NotificationMask = 0

End If
```

Remarks

Setting this property to a non-zero value, will enable the requested notifications. This is equivalent to calling the "RequestNotification". If NotificationMask is assigned to zero, notifications are disabled. This is equivalent to calling "CancelNotification"

Property

NotificationsActive

Type

long

Read-Only

Example (Visual Basic)

```
'' The Example assumes you have a form containing the DIC-OCX control named "DiiDevice".
'' See the following Chapter for information about how to use the OCX from Visual Basic

DicOcx.DeviceName = "Device0"

If Not DicOcx.DeviceOpened Then

        MsgBox "Unable To Open Device named Device0"

Else

        DicOcx.NotificationMask = &H2
                        '' Enables Signal Change Notifications

                        '' Check, if notifications were activated:
        If Not DicOcx.NotificationsActive Then
                MsgBox "This device does not support notifications"
        End If

                        '' Disable notifications again.
        DicOcx.NotificationMask = 0

End If
```

Remarks

This property indicates whether notifications were successfully enabled or disabled. If set to True, the OCX can fire "Notify" events at any time.
If set to False, the OCX does not fire "Notify" events.

- **Events**

In the current version, the DIC-OCX supports sending events (from report server) which correspond to notifications from the DII-DLL. (Server Side) Notifications can be send whenever a signal changes on the card, or if a counter drops to zero.

The DIC-OCX exposes one central event “Notify”. When a Notify event is fired, the following parameters are available:

```
Notify  (       ByVal NotificationMask As Long, _
                ByVal NotificationParam As Long, _
                ByVal DataBuffer As Variant, _
                ByVal BufferSize As Long
        )
```

Parameters

NotificationMask
A bitmask containing the notifications sent. In this mask, one or more bits corresponding to the previously setup NotificationMask property.

NotificationParam
Contains parameters for the notification. The interpretation of this parameter depends on the notification fired. See the description of the DII_NOTIFICATION_DATA for further explanation.

DataBuffer A variant containing a binary buffer of data samples accompanying the event. This is used for background sampling of data.

BufferSize The size of the buffer in bytes. (Not needed for Visual Basic).

Example (Visual Basic)

```
        '' The Example assumes you have a form containing the DIC-OCX control named "DiiDevice".
        '' See the following Chapter for information about how to use the OCX from Visual Basic


Private Sub Form_Load()

        DicOcx.DeviceName = "Device0"

        If Not DicOcx.DeviceOpened Then
                MsgBox "Unable To Open Device named Device0"
        Else
                DicOcx.NotificationMask = &H2
                                '' Enables Signal Change Notifications

                                '' Check, if notifications were activated:
                If Not DicOcx.NotificationsActive Then
                        MsgBox "This device does not support notifications"
                End If
        End If
End Sub

Private Sub Form_Unload()

                '' Disable notifications again.
        DicOcx.NotificationMask = 0
End Sub
```

```
Private Sub DicOcx_Notify(ByVal NotificationMask As Long, _
                          ByVal NotificationParam As Long, _
                          ByVal DataBuffer As Variant, _
                          ByVal BufferSize As Long)
                '
                ' This function is called whenever there is a notification
                ' sent by the device.

                ' Check the type of notification:

        If ((NotificationMask And &H2) <> 0) Then

                If ((NotificationParam And &H1) <> 0) Then
                        '' Signal 1 changed.
                End If

                If ((NotificationParam And &H2) <> 0) Then
                        '' Signal 2 changed.
                End If

                If ((NotificationParam And &H4) <> 0) Then
                        '' Signal 3 changed.
                End If

                If ((NotificationParam And &H8) <> 0) Then
                        '' Signal 4 changed.
                End If

                        '' ...
        End If

        If ((NotificationMask And &H20) <> 0) Then
                '
                ' It is a down-counter reached zero notification
                Beep
        End If
End Sub
```