Introducing the DAQFlex communication protocol

DAQFlex is a communication protocol used to develop data acquisition applications that can be deployed across multiple operating systems and custom embedded systems. The DAQFlex protocol greatly simplifies driver and application development. All DAQ operations are programmed through a common command interface composed of a consistent, cross-platform API, and an small footprint, open-source driver.

DAQFlex software includes the software API, FlexTest application, and example programs.

- Platform support and hardware requirements
- Installing the DAQFlex Software API Library
- Using the DAQFlexSoftware API
- DAQFlex API Reference
- DAQFlex Message Reference
- FlexText application
- C# and VB .NET Example programs
- Hardware Reference

You can run the DAQFlex software API on Windows, Mac, and Linux operating systems.

Help revision: 4.0

August, 2010

Platform support and hardware requirements

Platform support

You can run the DAQFlex communication protocol on a computer running one of the following operating systems and software:

- Microsoft Windows 7/Vista/XP (32-bit or 64-bit)
  - Windows 7
  - Windows Vista
  - Windows XP (SP2 or later)
  - Microsoft .NET® Framework 2.0 or later
  - WinUsb user-mode-driver
- Microsoft Windows CE
  - Microsoft .NET® Compact Framework 3.5 (X86 or XScale)
- Macintosh (32-bit or 64-bit)
  - MAC OS X
  - Leopard 10.5 or later
  - Mono Framework 2.0 or later
- Linux (32-bit or 64-bit)
  - Linux (2.4 kernel or later)
  - Mono Framework 2.0 or later
  - Libusb user-mode driver version 1.0.0.0

Hardware requirements

- Intel Pentium 4, 1 GHz or higher
- Minimum of 512 MG of RAM (1 GB or higher recommended)
- Video card with 128 MB memory
- Video display with 800 x 600 resolution or greater, and 256 colors or greater
- Microsoft-compatible mouse

Installing the DAQFlex software API library

The DAQFlex Software API operates with standard drivers for Windows, Mac, and Linux. Follow the procedure below specific to your operating system to install the DAQFlex software.

Windows 7, Windows Vista, Windows XP, Windows CE:

1. Run the Windows **Setup.exe** installer file located on the CD in the Windows folder.
2. Connect your DAQFlex device.
3. Run the **FlexTest.exe** test application from the Start menu. This application is an interactive C# console program that demonstrates how to use the DAQFlex software API.

   Additionally, you can build and run the C# or VB .NET example programs included in the installation using ExampleBuilder or Visual Studio (version 2005 or later).
4. Review the Hardware reference topic for the API components and messages supported by DAQFlex series hardware.

Mac OS X

1. Run the DAQFlex installer package located on the CD in the MAC_OSX folder.
2. Follow the installer instructions.
3. Connect your DAQFlex device.
4. Run the FlexTest application located in the /Applications/Measurement Computing/DAQFlex folder.
5. Review the Hardware reference topic for the API components and messages supported by DAQFlex series hardware.

Linux:

1. Using your Software/Package manager, verify that the **Mono framework** (version 2.4 or later) and the **libusb** user-mode driver are installed on your Linux system.

   If these versions aren't listed, information on installing, updating, or adding software repositories to your Software/Package manager can be found at the following links. Click here to go to the Mono web site. Click here to go to the libusb web site.
2. As a root user, create a symbolic link to the libusb-1.0 shared object file. For example:
   - $ ln -s /usr/lib/libusb-1.0.so.0/usr/lib/libusb-1.0.so
     (The actual file location may vary.)
3. Extract the files from the **DAQFlex-2.0.tar.gz** archive file on the DAQFlex software CD using an archive manager.
4. In a terminal window, set the current directory to DAQFlex/Source/DAQFlexAPI.
5. As a root user, run the following commands:
   - $ make
   - $ make install
6. Restart the system.
7. In a terminal window, set the current directory to DAQFlex/Source/DAQFlexTest.
8. As a root user, run the following commands:
   - $ make
   - $ make install
9. From a terminal window, run the **FlexTest** application using the following command:
   - $ flextest

   This application is an interactive C# console program that demonstrates how to create an application using the DAQFlex software API. Additionally, you can build and run the C# example programs included in the installation using MonoDevelop or the Mono command line interpreter.
10. Review the Hardware reference topic for the API components and messages supported by DAQFlex series hardware.

Using the DAQFlexSoftware API

This topic outlines how you program a device with DAQFlex.

The first step in programming a device with DAQFlex is to add a reference to the DAQFlex assembly to your project. In Visual Studio and MonoDevelop, this assembly is listed under the .NET tab of the Add Reference dialog as **DAQFlex API**. If your project is a C# project add the following statement to your source file:

```
using MeasurementComputing.DAQFlex;
```

The next step in programming a device with DAQFlex is to get a list of device names using the static method **GetDeviceNames()**:

C#

```
string[] deviceNames;

deviceNames = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno);
```

VB

```
Dim deviceNames As String()

deviceNames = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno)
```

GetDeviceNames gets the names of DAQFlex devices detected by the DAQFlex API. DeviceNameFormat is an enumeration that specifies the format of the returned values. This enumeration defines four different formats:

Member Name	Description
NameOnly	The returned values contain only the device name.
NameAndSerno	The return values contain the device name with the device serial number formatted as "DeviceName::SerialNumber".
NameAndID	The return values contains the device name with the device's user-defined ID formatted as "DeviceName::DeviceID".
NameSernoAndID	The return values contains the device name, the device serial number and the device's user-defined ID formatted as "DeviceName::SerialNumber::DeviceID".

Note: Each DAQFlex API method will throw an exception if an error occurs, and should be enclosed within a **Try/Catch** block.

The next step in programming a device with DAQFlex is to get a device object using the static method **CreateDevice()**:

C#

```
int deviceNumber = 0;

DaqDevice device;

string deviceName = deviceNames[deviceNumber];

device = DaqDeviceManager.CreateDevice(deviceName);
```

VB

```
Dim deviceNumber As Integer

Dim device As DaqDevice

Dim deviceName As String

deviceNumber = 0

deviceName = deviceNames(deviceNumber)

device = DaqDeviceManager.CreateDevice(deviceName)
```

Once you have a DaqDevice object you can use the **SendMessage()** method to program your DAQFlex device as shown below.

C#

```
DaqResponse response;

response = device.SendMessage("AI{0}:RANGE=BIP10V"); // set the input range for channel 0

response = device.SendMessage("?AI{0}:VALUE"); // read a single value from channel 0
```

VB

```
Dim response As DaqResponse

response = device.SendMessage("AI{0}:RANGE=BIP10V") ' set the input range for channel 0

response = device.SendMessage("?AI{0}:VALUE") ' read a single value from channel 0
```

The DaqResponse object contains a method for getting the response as a string, and a method for getting the response as a numeric. To get the response as a string, use the **ToString()** method:

C#

```
string value = response.ToString();
```

VB

```
Dim value As String

value = response.ToString()
```

To get the response as a numeric, use the **ToValue()** method:

C#

```
double value = response.ToValue();
```

VB

```
Dim value As Double

value = response.ToValue()
```

If the response does not contain a numeric value, ToValue() returns Double.NaN. When you no longer need the DaqDevice object, you can release it by calling the **ReleaseDevice()** method:

C#

```
DaqDeviceManager.ReleaseDevice(device);
```

VB

```
DaqDeviceManager.ReleaseDevice(device)
```

For more information

- Performing asynchronous single-point I/O — refer to Reading and writing software-paced I/O for example programs that demonstrate how to perform asynchronous single-point I/O using the DAQFlex Software API.
- Reading hardware-paced scan data — refer to Reading hardware-paced I/O for information on how to program a basic scan using the DAQFlex Software API.

Reading and writing software-paced I/O

The following examples demonstrate how to perform asynchronous single-point I/O using the DAQFlex Software API:

- Reading an analog input channel
- Writing to an analog output channel
- Reading a digital port
- Reading a digital bit
- Writing to a digital port
- Reading a counter input channel

Reading an analog input channel

C#

```
// Read the value of analog input channel 0
String[] Devices;
DaqDevice MyDevice;
DaqResponse Response;

try
{
   // Get a list of message-based DAQ devices
   Devices = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno);

   // Get a DaqDevice object for device 0
   MyDevice = DaqDeviceManager.CreateDevice(Devices[0]);

   // Send device messages using the DaqDevice object
   MyDevice.SendMessage("AI{0}:RANGE=BIP10V");
   MyDevice.SendMessage("AI:CAL=ENABLE");
   MyDevice.SendMessage("AI:SCALE=ENABLE");

   // Read and display the daq response
   Response = MyDevice.SendMessage("?AI{0}:VALUE");
   label1.Text = Response.ToString();
}
catch (Exception ex)
{
   // handle error
   label1.Text = ex.Message;
}
```

VB

```
' Read the value of analog input channel 0
Dim Devices As String ()
Dim MyDevice As DaqDevice
Dim Response As DaqResponse

Try
   ' Get a list of message-based DAQ devices
   Devices = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno)

   ' Get a DaqDevice object for device 0
   MyDevice = DaqDeviceManager.CreateDevice(Devices(0))

   ' Send device messages using the DaqDevice object
   MyDevice.SendMessage("AI{0}:RANGE=BIP10V")
   MyDevice.SendMessage("AI:CAL=ENABLE")
   MyDevice.SendMessage("AI:SCALE=ENABLE")

   ' Read and display the daq response
   Response = MyDevice.SendMessage("?AI{0}:VALUE")
   Label1.Text = Response.ToString()
Catch Ex As Exception
   ' handle error
   Label1.Text = Ex.Message()
End Try
```

Writing to an analog output channel

C#

```
// Write a value to analog output channel 0
String[] Devices;
DaqDevice MyDevice;

try
{
   // Get a list of message-based DAQ devices
   Devices = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno);

   // Get a DaqDevice object for device 0
   MyDevice = DaqDeviceManager.CreateDevice(Devices[0]);

   // Send device messages
   MyDevice.SendMessage("AO{0}:RANGE=BIP10V");
   MyDevice.SendMessage("AO:CAL=ENABLE");
   MyDevice.SendMessage("AO:SCALE=ENABLE");
   MyDevice.SendMessage("AO{0}:VALUE=2.53");
}
catch (Exception ex)
{
   // handle error
   label1.Text = ex.Message;
}
```

VB

```
' Write a value to analog output channel 0
Dim Devices As String()
Dim MyDevice As DaqDevice

Try
   ' Get a list of message-based DAQ devices
   Devices = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno)

   ' Get a DaqDevice object for device 0
   MyDevice = DaqDeviceManager.CreateDevice(Devices(0))

   ' Send device messages
   MyDevice.SendMessage("AO{0}:RANGE=BIP10V")
   MyDevice.SendMessage("AO:CAL=ENABLE")
   MyDevice.SendMessage("AO:SCALE=ENABLE")
   MyDevice.SendMessage("AO{0}:VALUE=2.53")
Catch Ex As Exception
   ' handle error
   Label1.Text = Ex.Message
End Try
```

Reading a digital bit

C#

```
// Read the value of digital port 0, bit 0
String[] Devices;
DaqDevice MyDevice;
DaqResponse Response;

try
{
   // Get a list of message-based DAQ devices
   Devices = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno);

   // Get a DaqDevice object for device 0
   MyDevice = DaqDeviceManager.CreateDevice(Devices[0]);

   // Read and display the daq response
   MyDevice.SendMessage("DIO{0/0}:DIR=IN");
   Response = MyDevice.SendMessage("?DIO{0/0}:VALUE");
   label1.Text = Response.ToString();
}
catch (Exception ex)
{
   // handle error
   label1.Text = ex.Message;
}
```

VB

```
' Read the value of digital port 0, bit 0
Dim MyDevice As DaqDevice
Dim Response As DaqResponse
Dim Devices As String ()

Try
   ' Get a list of message-based DAQ devices
   Devices = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno)

   ' Get a DaqDevice object for device 0
   MyDevice = DaqDeviceManager.CreateDevice(Devices(0))

   ' Read and display the daq response
   MyDevice.SendMessage("DIO{0/0}:DIR=IN")
   Response = MyDevice.SendMessage("?DIO{0/0}:VALUE")
   Label1.Text = Response.ToString()
Catch Ex As Exception
   ' handle error
   Label1.Text = Ex.Message
End Try
```

Reading a digital port

C#

```
// Read the value of digital port 0
String[] Devices;
DaqDevice MyDevice;
DaqResponse Response;

try
{
   // Get a list of message-based DAQ devices
   Devices = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno);

   // Get a DaqDevice object for device 0
   MyDevice = DaqDeviceManager.CreateDevice(Devices[0]);

   // Send device messages
   MyDevice.SendMessage("DIO{0}:DIR=IN");

   // Read and display the daq response
   Response = MyDevice.SendMessage("?DIO{0}:VALUE");
   Label1.Text = Response.ToString();
}
catch (Exception ex)
{
   // handle error
   label1.Text = ex.Message;
}
```

VB

```
' Read the value of digital port 0
Dim MyDevice As DaqDevice
Dim Response As DaqResponse
Dim Devices As String ()

Try
   ' Get a list of message-based DAQ devices
   Devices = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno)

   ' Get a DaqDevice object for device 0
   MyDevice = DaqDeviceManager.CreateDevice(Devices(0))

   ' Send device messages
   MyDevice.SendMessage("DIO{0}:DIR=IN")

   ' Read and display the daq response
   Response = MyDevice.SendMessage("?DIO{0}:VALUE")
   Label1.Text = Response.ToString()
Catch Ex As Exception
   ' handle error
   Label1.Text = Ex.Message
End Try
```

Writing to a digital port

C#

```
// Write a value to digital port 0
String[] Devices;
DaqDevice MyDevice;
DaqResponse Response;

try
{
   // Get a list of message-based DAQ devices
   Devices = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno);

   // Get a DaqDevice object for device 0
   MyDevice = DaqDeviceManager.CreateDevice(Devices[0]);

   // Send device messages
   MyDevice.SendMessage("DIO{0}:DIR=OUT");
   MyDevice.SendMessage("DIO{0}:VALUE=128");
   label1.Text = response.ToString();
}
catch (Exception ex)
}
   // handle error
   label1.Text = ex.Message;
}
```

VB

```
' Write a value to digital port 0
Dim MyDevice As DaqDevice
Dim Response As DaqResponse
Dim Devices As String ()

Try
   ' Get a list of message-based DAQ devices
   Devices = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno)

   ' Get a DaqDevice object for device 0
   MyDevice = DaqDeviceManager.CreateDevice(Devices(0))

   ' Send device messages
   MyDevice.SendMessage("DIO{0}:DIR=OUT")
   Response = MyDevice.SendMessage("DIO{0}:VALUE=128")
   Label1.Text = Response.ToString
Catch Ex As Exception
   ' handle error
   Label1.Text = Ex.Message
End Try
```

Reading a counter input channel

C#

```
// Read counter channel 0
String[] Devices;
DaqDevice MyDevice;
DaqResponse Response;

try
{
   // Get a list of message-based DAQ devices
   Devices = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno);

   // Get a DaqDevice object for device 0
   MyDevice = DaqDeviceManager.CreateDevice(Devices(0));

   // Start the counter
   MyDevice.SendMessage("CTR{0}:VALUE=0");
   MyDevice.SendMessage("CTR{0}:START");

   // Read and display the daq response
   for(int i = 1;i<=10;i++)
   {
       System.Threading.Thread.Sleep(750);
       Response = MyDevice.SendMessage("?CTR{0}:VALUE");
       label1.Text = Response.ToString();
       Application.DoEvents();
   }

   // Stop the counter
   MyDevice.SendMessage("CTR{0}:STOP");
}
catch (Exception ex)
{
   // handle error
   label1.Text = ex.Message;
}
```

VB

```
' Read counter channel 0
Dim MyDevice As DaqDevice
Dim Response As DaqResponse
Dim Devices As String ()

Try
   ' Get a list of message-based DAQ devices
   Devices = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno)

   ' Get a DaqDevice object for device 0
   MyDevice = DaqDeviceManager.CreateDevice(Devices(0))

   Dim I As Integer

   ' Start the counter
   MyDevice.SendMessage("CTR{0}:VALUE=0")
   MyDevice.SendMessage("CTR{0}:START")

   ' Read and display the daq response
   For I = 1 To 10
       System.Threading.Thread.Sleep(750)
       Response = MyDevice.SendMessage("?CTR{0}:VALUE")
```

```
        Label1.Text = Response.ToString()
        Application.DoEvents()
    Next

    ' Stop the counter
    MyDevice.SendMessage("CTR{0}:STOP")
Catch Ex As Exception
    ' handle error
    Label1.Text = Ex.Message
End Try
```

Reading hardware-paced I/O

The basic approach to programming analog input scans is to set the device's scan properties, send the **START** command and call the **ReadScanData**() method. The following examples show how to program a basic scan.

C#

```
try
{
   double[,] scanData;
   string[] names = DaqDeviceManager.GetDeviceNames
   (DeviceNameFormat.NameAndSerno);
   DaqDevice device = DaqDeviceManager.CreateDevice(names[0]);

   device.SendMessage("AISCAN:LOWCHAN=0");
   device.SendMessage("AISCAN:HIGHCHAN=0");
   device.SendMessage("AISCAN:RATE=1000");
   device.SendMessage("AISCAN:SAMPLES=5000");
   device.SendMessage("AISCAN:START");

   scanData = device.ReadScanData(5000, 0);

}
catch (Exception ex)
{
   Console.WriteLine(ex.Message);
}
```

VB

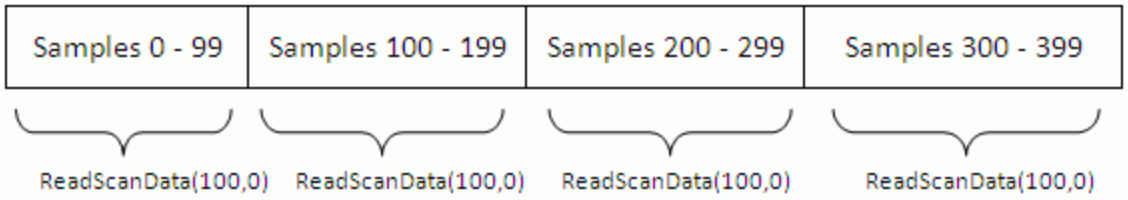
```
Try
   Dim ScanData As Double(,)
   Dim Names As String()
   Dim Device As DaqDevice

   Names = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno)
   Device = DaqDeviceManager.CreateDevice(Names(0))

   Device.SendMessage("AISCAN:LOWCHAN=0")
   Device.SendMessage("AISCAN:HIGHCHAN=0")
   Device.SendMessage("AISCAN:RATE=1000")
   Device.SendMessage("AISCAN:SAMPLES=5000")
   Device.SendMessage("AISCAN:START")

   ScanData = Device.ReadScanData(5000, 0)

Catch ex As Exception
   Console.WriteLine(ex.Message)
End Try
```

ReadScanData parameters

The first parameter to the **ReadScanData** method is the number of samples to read. The second parameter is a time out value in milli-seconds. A value of 0 indicates no timeout specified.

The ReadScanData method is synchronous, and will return when the number of requested samples are available for reading. When the number of requested samples are available, the DAQFlex API will copy the requested number of samples from an internal buffer to a new array of data. The DAQFlex API keeps track of the buffer index so that multiple calls to ReadScanData always return contiguous data.

Internal Buffer



An alternative method for reading scan data is to enable a **user-defined callback method**. When you enable a callback method, the DAQFlex API invokes your user-defined method when a specified number of samples are available for reading, when a scan completes, or if a scan error occurs. This is done using the **EnableCallback** method as shown below.

C#

```
Device.EnableCallback(callbackMethod, callbackType, callbackCount);
```

VB

```
Device.EnableCallback(Addressof CallbackMethod, CallbackType, CallbackCount)
```

The callbackMethod is the name of the method that will be invoked by the DAQFlex API. The callbackMethod is a class method that must have the following form:

C#

```
void CallbackMethod(ErrorCodes errorCode, CallbackType callbackType, object callbackData)
```

VB

```
Sub CallbackMethod(ByVal errorCode As ErrorCodes, ByVal callbackType As CallbackType, _ ByVal
callbackData As Object)
```

The callbackType is an enumeration that defines when the callback method will be invoked.

CallbackType

Member Name	Description
OnDataAvailable	Specifies that the callback method will be invoked when a specified number of samples becomes available for reading.
OnInputScanComplete	Specifies that the callback method will be invoked when a finite scan has complete or when a continuous scan is stopped.
OnInputScanError	Specifies that the callback method will be invoked when an input scan error occurs.

Only one callback method can be specified for each callback type. When the callback type is set to OnDataAvailable, set the callbackData parameter to the number of samples you wish to receive in the callback method. When the callback type is set to OnInputScanComplete or OnInputScanError, set the callbackData parameter to null or Nothing.

The following are examples of reading scan data using a callback method:

C#

```
try
{
   double[,] scanData;

   string[] names = DaqDeviceManager.GetDeviceNames
   (DeviceNameFormat.NameAndSerno);
   DaqDevice device = DaqDeviceManager.CreateDevice(names[0]);

   device.EnableCallback(OnReadScanData, CallbackType.OnDataAvailable, 1000);
   device.EnableCallback(OnReadScanData, CallbackType.OnScanComplete, null);

   device.SendMessage("AISCAN:LOWCHAN=0");
   device.SendMessage("AISCAN:HIGHCHAN=0");
   device.SendMessage("AISCAN:RATE=1000");
   device.SendMessage("AISCAN:SAMPLES=5000");
   device.SendMessage("AISCAN:START");
}
catch (Exception ex)
```

```
{
    Console.WriteLine(ex.Message);
}

protected void OnReadScanData(ErrorCodes errorCode, CallbackType callbackType,
object callbackData)
{
    try
    {
        int availableSamples = (int)callbackData;
        double[,] scanData = device.ReadScanData(availableSamples, 0);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

VB

```
Try
    Dim ScanData As Double(,)
    Dim Names As String()
    Dim Device As DaqDevice

    Names = DaqDeviceManager.GetDeviceNames(DeviceNameFormat.NameAndSerno)
    Device = DaqDeviceManager.CreateDevice(Names(0))

    Device.EnableCallback(AddressOf OnReadScanData, CallbackType.OnDataAvailable,
    1000)
    Device.EnableCallback(AddressOf OnReadScanData, CallbackType.OnScanComplete,
    Nothing)

    Device.SendMessage("AISCAN:LOWCHAN=0")
    Device.SendMessage("AISCAN:HIGHCHAN=0")
    Device.SendMessage("AISCAN:RATE=1000")
    Device.SendMessage("AISCAN:SAMPLES=5000")
    Device.SendMessage("AISCAN:START")
Catch ex As Exception
    Console.WriteLine(ex.Message)
End Try

Protected Sub ReadScanData(ByVal errorCode As ErrorCodes, ByVal callbackType As
CallbackType, _ ByVal callbackData As Object)

Try
    Dim AvailableSamples As Integer
    Dim ScanData As Double(,)

    AvailableSamples = DirectCast(callbackData, Integer)
    ScanData = Device.ReadScanData(AvailableSamples, 0)
Catch ex As Exception
    Console.WriteLine(ex.Message)
End Try
```

DAQFlex API Reference

The DAQFlex Software API is an open source library that implements a simple message-based protocol consisting of text-based commands, or messages. The API is written in C#, is designed for cross platform portability, and does not require a separate configuration utility or a configuration file.

The protocol defines a number of DAQ *components*. The components are device elements which encapsulate a different DAQ subsystem, such as analog input or digital I/O. Each message that is sent to a device pertains to a specific DAQ component. The messages are used to do the following:

- program a DAQ device
- read data from a DAQ device
- write data to a DAQ device
- read information about a DAQ device, such as the maximum scan rate or support for an external clock

The messages are identical for all devices that support the protocol, eliminating the need to learn device-specific commands.

DAQFlex API classes

The DAQFlex API contains two classes:

- DaqDeviceManager class
- DaqDevice class

Each class includes multiple methods.

DaqDeviceManager class

The **DaqDeviceManager** class includes the following methods:

- GetDeviceNames() – gets a list of devices that support the message-based protocol.
- CreateDevice() – creates a DaqDevice object, which contains the methods used to communicate with a DAQ device.
- ReleaseDevice() – frees the resources associated with a DaqDevice object.

DaqDeviceManager.GetDeviceNames() method

Gets a list of DAQ devices that support the message-based protocol.

Declaration

C#: static string[] GetDeviceNames(DeviceNameFormat format);

VB: Shared Function GetDeviceNames(ByVal format As DeviceNameFormat) As String()

Parameter

format

The format to use for a device name. This parameter is a DeviceNameFormat enumeration. The enumeration values and the format of the return strings are listed below:

Value	Return string format
NameOnly	"Device name"
NameAndSerno	"Device name::Device serial number"
NameAndID	"Device name::Device ID"
NameSernoAndID	"Device name::Device serial number::Device ID"

Return value

An array of strings containing the device names of all DAQ devices that support the message-based protocol.

Remarks

- The values contained in the array can be used to create a **DaqDevice object** for the device that you want to program.

  The DaqDevice object is created using the DaqDeviceManager's **CreateDevice** static method.

  With the DaqDevice object, all DAQ operations are configured using one API method called **SendMessage()** rather than using multiple operation-specific methods.

- The **NameOnly** format is not useful if multiple devices of the same type are connected, since the application won't be able to differentiate between one device and the other. If you are using multiple devices of the same type, then use one of the other formats.

  If any of the devices in use do not have an ID assigned, the formats that use the ID (**NameAndID** and **NameSernoAndID**) will generate an error if used with the DaqDeviceManager.CreateDevice() method. Use one of the other formats (**NameOnly** or **NameAndSerno**) if you are using devices without an ID assigned.

DaqDeviceManager.CreateDevice() method

Creates a DaqDevice object. The DaqDevice object contains the methods used to configure, read data from, or write data to a device. With the DaqDevice object, all DAQ operations are configured using one API method called DaqDevice.SendMessage() rather than using multiple operation-specific methods. SendMessage() takes a single parameter called *message*. This parameter is a text-based command that the DAQ device parses to configure a particular operation.

Declaration

C#: static DaqDevice CreateDevice(string deviceName);

VB: Shared Function CreateDevice(ByVal deviceName As String) As DaqDevice

Parameter

deviceName

One of the device names returned by the DaqDeviceManager.GetDevicenames() method.

Return value

An instance of a **DaqDevice object**.

Remarks

- Depending on the DeviceNameFormat, the CreateDevice() method creates a DaqDevice object for the device whose name, name and serial number, name and id, or name, serial number and id are contained in the deviceName parameter.
- The resources associated with the DaqDevice object can be freed by calling the ReleaseDevice() method.
- The CreateDevice() method can only be called once for a specific device, unless the ReleaseDevice() method is called.

  If CreateDevice() is called more than once for a specific device without calling ReleaseDevice(), the DaqDevice object **throws an exception**, indicating that a driver handle has already been created for the device.

  Refer to the sample code below:

C#

```
try
{
   MyDevice = DaqDeviceManager.CreateDevice(deviceName);
}
catch (Exception ex)
{
   // handle exception
}
```

VB

```
Try
   MyDevice = DaqDeviceManager.CreateDevice(deviceName)
Catch Ex As Exception
   ' handle exception
End Try
```

DaqDeviceManager.ReleaseDevice() method

Frees the resources associated with a DaqDevice object.

Declaration

C#: static void ReleaseDevice(DaqDevice device);

VB: Shared Sub ReleaseDevice(ByVal device As DaqDevice)

Parameter

device

A DaqDevice object created by the CreateDevice() method.

DaqDevice class

The **DaqDevice** class includes the following methods:

- SendMessage() – takes a single text-based command that the DAQ device parses to configure a particular operation.
- ReadScanData() – reads hardware-paced scan data.
- EnableCallback() – enables a user-defined callback method to be invoked when a certain condition is met. This method is used in conjunction with input scan operations.
- DisableCallback() – the condition that invokes the callback method.
- GetErrorMessage() – gets the error message associated with the error code that is passed to the user-defined callback.

DaqDevice.SendMessage()

Configures DAQ operations. This method takes a single text-based command that the DAQ device parses to configure a particular operation.

Declaration

C#: DaqResponse SendMessage (string message);

VB: Function SendMessage (ByVal Message as String) As DaqResponse

Parameter

Message

The text-based message to send to the device.

Return value

The device response as an instance of a **DaqResponse object**.

Remarks

- *Message* is a string containing a text-based command supported by the device, and *Response* is a DaqResponse object containing the device's response.
- The DaqResponse object includes two methods:
  - **ToString()**: gets the response as a string, for example "*AI{0}:VALUE=139*".
  - **ToValue()**: gets the response as a numeric value, for example "*139.0000*".
- All messages provide a string response, but not all messages provide a numeric response. For those messages that do not provide a numeric response, the numeric value is set to NaN (not a number).
- The ToString method has additional overloads that accept formatting parameters. The overloads are ToString(string format), ToString(IFormatProvider provider) and ToString(string format, IFormatProvider provider). The overloads can be used to format the numeric part of a response, if present. If the response does not contain a numeric, these overloads are ignored.
- If an error occurs while sending a message to a device, the SendMessage method will **throw an exception** rather than returning an error code. This means the application should encapsulate calls to SendMessage within a try/catch block.

  Refer to the following sample code.

C#

```
try
{
   DaqResponse response = MyDevice.SendMessage(message);
   label1.Text = response.ToString();
}
Catch (Exception ex)
{
   // handle exception
   label1.Text = ex.Message;
}
```

VB

```
Try
   DaqResponse Response = MyDevice.SendMessage(Message)
   Label1.Text = Response.ToString()
Catch Ex As Exception
   ' handle exception
   Label1.Text = Ex.Message;
End Try
```

DaqDevice.ReadScanData() method

Reads data for a scan operation.

Declaration

C#: double[] ReadScanData(int samples);

VB: Function ReadScanData(ByVal samples As Integer) As Double()

Parameter

samples

The number of samples per channel to read.

Return value

An array of data samples read from the device.

Remarks

- The DAQFlex library always performs scan operations in the background, but ReadScanData() always runs in the foreground. When called, ReadScanData() returns control to the application that called it when the number of samples requested has been read. The DAQFlex library manages all memory allocation and array indexing so the application doesn't have to.

DaqDevice.EnableCallback()

Enables a user-defined callback method to be invoked when a certain condition is met.

Declaration

C#: void EnableCallback(ErrorCodes errorCode, CallbackType callbackType, Object callbackData

VB: Sub EnableCallback(ByVal errorCode as ErrorCodes, ByVal callback as InputScanCallbackDelegate, ByVal callbackType As CallbackType, ByVal callbackData As Object)

Parameter

errorCode

The error code passed to the callback method.

callbackType

The condition that invokes the callback method. Callback types are defined by the CallbackType enumeration. The supported types are:

- CallbackType.OnDataAvailable – Invokes the callback method when the number of samples available for reading data is greater than or equal to the number of samples specified by the callbackData parameter.
- CallbackType.OnInputScanComplete – Invokes the callback method when a finite input scan completes or a continuous scan is stopped.
- CallbackType.OnInputScanError – Invokes the callback method when a scan error has occurred.

callbackData

When the callbackType parameter is set to CallbackType.OnDataAvailable, set the callbackData parameter to the number of samples per channel to acquire before invoking the user-defined callback method.

When the callbackType parameter is set to CallbackType.OnInputScanComplete or OnInputScanError, set the callbackData parameter to null or Nothing.

Return value

The value of callbackType.

Remarks

- This method is used in conjunction with input scan operations.
- EnableCallback may be called once for each callback type. If it is called more than once for the sample callback type, a DaqException is thrown.

DaqDevice.DisableCallback()

Disables the invocation of the user-defined callback method associated with the callback type.

Declaration

C#: void DisableCallback(CallbackType callbackType)

VB: Sub DisableCallback(ByVal callbackType As CallbackType)

Parameter

callbackType

The callback type to disable.

DaqDevice.GetErrorMessage()

Gets the error message associated with the error code that is passed to the user-defined callback.

Declaration

C#: string GetErrorMessage(ErrorCodes errorCode)

VB: Function GetErrorMessage(ByVal errorCode As ErrorCodes)

Parameter

errorCode

The error code passed to the callback method.

Return value

The error message passed to the user-defined callback method.

DAQFlex Message Reference

The API messages that you send to a DAQFlex supported device are text-based commands. Each message pertains to a specific *DAQ component*. A DAQ component is a device element that encapsulates a DAQ subsystem which has multiple properties or commands associated with it.

DAQFlex components

The DAQFlex API defines the following DAQ components:

- DEV — encapsulates device-level operations
- AI — encapsulates single-point analog input operations
- AISCAN — encapsulates analog input scanning operations
- AITRIG — encapsulates analog input triggering operations
- AO — encapsulates single-point analog output operations
- AOSCAN — encapsulates analog output scanning operations
- AOTRIG — encapsulates analog output triggering operations
- DIO — encapsulates digital I/O operations
- CTR — encapsulates counter input operations
- TMR — encapsulates timer output operations

Each component has one or more properties associated with it. Each property supports one or more text-based commands, or *messages*. These messages are used to communicate with DAQflex-supported hardware.

DAQFlex messages

DAQFlex supports two types of messages:

- Device programming messages configure or retrieve a device property value.
- Device reflection messages retrieve information about a device capability, such as the maximum scan rate or support for an external clock.

Device programming messages

Device programming messages are used to get and set device properties. Programming messages that query a device property always start with the **?** character.

Click on a component below for the string messages, device responses, and property values supported by the component.

Components for programming a device	Description
AI	Sets and gets property values for analog input channels.
AISCAN	Sets and gets property values when scanning analog input channels.
AITRIG	Sets and gets analog input trigger property values.
AO	Sets and gets property values for analog output channels
AOSCAN	Sets and gets property values when scanning analog output channels.
AOTRIG	Sets and gets analog output trigger property values.
DEV	Sets and gets device property values.
DIO	Sets and gets property values for digital I/O channels.
CTR	Sets and gets property values for counter channels.
TMR	Sets and gets property values for timer output channels.

AI

Sets and gets property values for analog input channels.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

CHMODE, RANGE, VALUE, SENSOR, CJC, STATUS, SLOPE, OFFSET, CAL, SCALE

- Get the number of analog input channels on a device.

Message	"?AI"	
Response	"AI=*value*"	
	value	The number of A/D channels on the device.
Example	"AI=16"	

CHMODE

- Set the analog input mode to single-ended or differential.

Message	"AI:CHMODE=*value*"	
Response	"AI:CHMODE"	
	value	SE, DIFF

- Get the input mode that is set for the analog inputs.

Message	"?AI:CHMODE"	
Response	"AI:CHMODE=*value*"	
	value	SE, DIFF
Example	"AI:CHMODE=SE"	

RANGE

- Set the range value for a specified channel.

Message	"AI{*ch*}:RANGE=*value*"	
Response	"AI{*ch*}:RANGE"	
	ch	The channel number
	value	BIP20V (±20 volts) BIP10V (±10 volts) BIP5V (±5 volts) BIP4V (±4 volts) BIP2PT5V (±2.5 volts) BIP2V (±2 volts) BIP1PT25V (±1.25 volts) BIP1V (±1 volts) BIP73.125E-3V (±0.073125 volts) BIP146.25E-3V (±0.14625 volts)
Example	"AI{0}RANGE=BIP10V"	
Note	If RANGE is not specified, the device's power up default value is used.	

- Get the range value for a specified channel.

Message	"?AI{*ch*}:RANGE"	
Response	"AI{ch}:RANGE=*value*"	
	ch	The channel number
	value	The range value
Example	"?AI{0}:RANGE"	

VALUE

- Get the calibrated A/D count of a specified channel.

Message	"?AI{ch}:VALUE"	
Response	"AI{*ch*}:VALUE=*value*"	
	ch	The channel number
	value	The calibrated A/D count
Example	"?AI{0}:VALUE"	

- Get the A/D value in the specified format.

Message	"?AI{ch}:VALUE/format"[1]	
Response	"AI{*ch*}:VALUE/*format*=*value*"	
	ch	The channel number
	format	The data format of the measurement: ■ RAW: returns uncalibrated A/D counts ■ VOLTS: returns a calibrated A/D voltage ■ DEGC: returns a calibrated temperature value in °C ■ DEGF: returns a calibrated temperature value in °F ■ KELVIN: returns a calibrated temperature value in Kelvin
	value	The A/D value
Example	"?AI{0}:VALUE/DEGC"	
Note	This message is processed by the DAQFlex Software library, and is not sent to the device.	

SENSOR

- Set the thermocouple sensor type.

Message	"AI{*ch*}:SENSOR=TC/*value*"	
Response	"AI:{*ch*}:SENSOR"	
	value	The thermocouple type
Example	"AI{0}:SENSOR=TC/K"	
Note	Thermocouple types B, E, J, K, N, R, S, and T are supported.	

- Get the thermocouple sensor type.

Message	"?AI{*ch*}:SENSOR"	
Response	"AI:{*ch*}:SENSOR=TC/*value*"	
	value	The thermocouple type
Example	"?AI{0}:SENSOR"	
Note	Thermocouple types B, E, J, K, N, R, S, and T are supported.	

CJC

- Get the CJC value in the specified format.

Message	"?AI{*ch*}:CJC/*format*"	
Response	"AI{*ch*}:CJC/*format*=*value*"	
	format	DEGC, DEGF, KELVIN
	value	The measured temperature
Example	"?AI{0}:CJC/DEGC"	

STATUS

- Get the current ADC status of the AI operation.

Message "?AI:STATUS"

Response "AI:STATUS=*value*"

value BUSY, ERROR, or READY

SLOPE

- Get the calibration slope coefficient for the specified channel.

Message "?AI{*ch*}:SLOPE"

Response "AI{*ch*}:SLOPE=*value*"

ch The channel number

value The calibration slope

Example "?AI{0}:SLOPE"

OFFSET

- Get the calibration offset coefficient for the specified channel.

Message "?AI{*ch*}:OFFSET"

Response "AI{*ch*}:OFFSET=*value*"

ch The channel number

value The calibration offset

Example "?AI{0}:OFFSET"

CAL

- Enable or disable calibration of all A/D channels.

Message "AI:CAL=*value*"

Response "AI:CAL"

value ENABLE or DISABLE

Example "AI:CAL=ENABLE"

Note This message is processed by the DAQFlex Software library, and is not sent to the device.

- Get a value indicating whether the calibration coefficients will be applied to the raw A/D data.

Message "?AI:CAL"

Response "AI:CAL=*value*"

value ENABLE or DISABLE

Example "?AI:CAL"

Note This message is processed by the DAQFlex Software library, and is not sent to the device.

SCALE

- Enable or disable scaling of A/D channels.

Message "AI:SCALE=*value*"

Response "AI:SCALE"

value ENABLE or DISABLE

Example "AI:SCALE=ENABLE"

Note This message is processed by the DAQFlex Software library, and is not sent to the device.

- Get a value indicating whether scaling will be applied to the A/D channels.

Message "?AI:SCALE"

Response	"AI:SCALE=*value*"
	value ENABLE or DISABLE
Note	This message is processed by the DAQFlex Software library, and is not sent to the device.

Working with the CAL and SCALE properties

The ENABLE/DISABLE setting of the CAL and SCALE properties affect the kind of data that is returned:

- CAL=DISABLE, SCALE=DISABLE

  If CAL and SCALE are both disabled, the data returned will be raw A/D integer values within the range of 0 to $2^{resolution}$ -1 of the device. If the calibration factors are stored on the device and applied to the data by the application software, the data range may be limited to well within these values.

- CAL=ENABLE, SCALE=DISABLE

  When CAL is enabled and SCALE is disabled, the format of the analog data returned will depend on the type of calibration implemented. If the calibration factors are stored on the device and applied to the data by the application software, the data will be floating point values, not integer values, and may exceed the theoretical limits and include negative values and values above $2^{resolution}$ -1.

- SCALE=ENABLE

  When SCALE is enabled, scaled floating point values are returned. The limits of the data will depend on the implementation of calibration, as described above. Data range limits may be a small percentage less than or greater than the full scale range selected for devices for which the calibration factors are stored on the device and applied to the data by the application software.

AISCAN

Sets and gets property values when scanning analog input channels.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

XFRMODE, RATE, SAMPLES, BUFSIZE, EXTPACER, HIGHCHAN, LOWCHAN, DEBUG, TRIG, RANGE, QUEUE, START, STOP, STATUS, COUNT, CAL, SCALE

XFRMODE

- Set the transfer mode for analog input scan data.

Message	"AISCAN:XFRMODE=*value*"
Response	"AISCAN:XFRMODE"
	value SINGLEIO or BLOCKIO
Example	"AISCAN:XFRMODE=BLOCKIO"

- Get the transfer mode that is set for the scan.

Message	"?AISCAN:XFRMODE"
Response	"AISCAN:XFRMODE=*value*"
	value SINGLEIO, BLOCKIO

RATE

- Set the A/D sampling rate in Hz for each channel.

Message	"AISCAN:RATE=*value*"
Response	"AISCAN:RATE"
	value 0 to N
Example	"AISCAN:RATE=1000"
Note	■ If *value* is less than the device's minimum sampling rate, then the minimum rate is used. If *value* is greater than the device's maximum sampling rate, then the device's maximum sampling rate is used. ■ You should check the actual scan rate set using the "?AISCAN:RATE" query after setting the RATE.

- Get the A/D sampling rate in Hz for each channel.

Message	"?AISCAN:RATE"
Response	"AISCAN:RATE=*value*"
	value A value between the device's minimum and maximum rate.
Note	■ The *value* returned may not match the value requested using the "AISCAN:RATE=*value*" message due to device limitations. ■ If the *value* returned is at or near the device's maximum sampling rate, then sending other messages to the device should be kept to a minimum. Otherwise, there is a potential for a data overrun to occur. A data overrun occurs when the device fills its buffer with data faster than it is read back.

SAMPLES

- Set the number of samples/channel to acquire in the scan.

Message	"AISCAN:SAMPLES=*value*"
Response	"AISCAN:SAMPLES"
	value 0 to N
Example	"AISCAN:SAMPLES=1000"
Note	A value of 0 results in a continuous scan.

- Get the number of samples/channel to acquire in the scan.

Message "?AISCAN:SAMPLES"

Response "AISCAN:SAMPLES=*value*"

value 0 to N

Note A value of 0 results in a continuous scan.

BUFSIZE

- Set the size in bytes of the buffer to be used for AISCAN.

Message "AISCAN:BUFSIZE=*value*"

Response "AISCAN:BUFSIZE"

value The size in bytes of the buffer.

Example "AISCAN:BUFSIZE=131072"

Note
- The default buffer size 1024000 bytes. This should be sufficient for most applications. The actual buffer size will always be an integer multiple of the device's maximum packet size.
- If this value is set, it should be at least the (number of bytes per sample) x (number of channels) x (sample count) for finite mode, and twice this value for continuous mode.

- Get the size of the buffer used for AISCAN.

Message "?AISCAN:BUFSIZE"

Response "AISCAN:BUFSIZE=*value*"

value The size in bytes of the buffer.

EXTPACER

- Set the configuration of the device's external pacer pin.

Message "AISCAN:EXTPACER=*value*"

Response "AISCAN:EXTPACER"

value ENABLE (for most devices).

Example "AISCAN:EXTPACER=ENABLE"

Note
- Some devices support ENABLE/MASTER, ENABLE/SLAVE, ENABLE/GSLAVE, DISABLE/MASTER, or DISABLE/SLAVE.

  For devices which do not support a master and slave configuration, the /MASTER and /SLAVE designation is ignored.

  For devices which do not support disabling of the pacer or SYNC input (for example, the terminal is always enabled as either input or output), the DISABLE designation is invalid.

  When the optional portion of the message is omitted, the default is SLAVE.
- Set to *ENABLE* if the device is paced using a continuous clock source, such as a generator. In this mode, the first clock pulse after setting up the scan is ignored in order to ensure adequate setup time for the first conversion.

  Set to *ENABLE/GSLAVE* if the device is paced from a DAQFlex Series device. In this mode, the first clock pulse after setting up the scan is held off to ensure adequate setup time for the first conversion. No pulses are ignored.
- When the external pacer is enabled, the AISCAN:RATE property should be set to approximately what the external pacer rate will be, because internal transfer sizes are calculated using the rate and channel count.

- Gets the configuration of the device's external pacer pin.

Message "?AISCAN:EXTPACER"

Response "AISCAN:EXTPACER=*value*"

ch The channel number.

value ENABLE (for most devices).

Note Some devices support ENABLE/MASTER, ENABLE/SLAVE, ENABLE/GSLAVE, DISABLE/MASTER, or

DISABLE/SLAVE.

HIGHCHAN

- Set the last channel to include in the hardware-paced scan operation.

Message "AISCAN:HIGHCHAN=*value*"

Response "AISCAN:HIGHCHAN"

value The channel number.

Example "AISCAN:HIGHCHAN=3"

- Get the last channel to include in the hardware-paced scan operation.

Message "?AISCAN:HIGHCHAN"

Response "AISCAN:HIGHCHAN=*value*"

value The channel number.

LOWCHAN

- Set the first channel to include in the hardware-paced scan operation.

Message "AISCAN:LOWCHAN=*value*"

Response "AISCAN:LOWCHAN"

value The channel number.

Example "AISCAN:LOWCHAN=0"

- Get the first channel to include in the hardware-paced scan operation.

Message "?AISCAN:LOWCHAN"

Response "AISCAN:LOWCHAN=*value*"

value The channel number.

DEBUG

- Enable or disable the debug feature.

Message "AISCAN:DEBUG=*value*"

Response "AISCAN:DEBUG"

value ENABLE, DISABLE

Example "AISCAN:DEBUG=ENABLE"

Note When the debug feature is enabled, the data returned by ReadScanData() is an incrementing count from 0 to the maximum value of the device's A/D. The count is reset to 0 when the maximum value is reached. For 16-bit A/Ds the maximum value is 65535.

- Get the debug status.

Message "?AISCAN:DEBUG"

Response "AISCAN:DEBUG=*value*"

ch The channel number.

value ENABLE, DISABLE

TRIG

- Enable or disable the trigger.

Message "AISCAN:TRIG=*value*"

Response "AISCAN:TRIG"

value ENABLE, DISABLE

Example "AISCAN:TRIG=ENABLE"

Note If not set, TRIG is set to DISABLE by default.

- Get the trigger status.

Message "?AISCAN:TRIG"

Response "AISCAN:TRIG=*value*"

value ENABLE, DISABLE

RANGE

- Set the range for the analog input channels to be scanned.

Message "AISCAN:RANGE=*value*"

Response "AISCAN:RANGE"

value Range value:

- BIP20V (±20 volts)
- BIP10V (±10 volts)
- BIP5V (±5 volts)
- BIP4V (±4 volts)
- BIP2.5V (±2.5 volts)
- BIP2V (±2 volts)
- BIP1.25V (±1.25 volts)
- BIP1V (±1 volt)

Example "AISCAN:RANGE=BIP20V"

Note If RANGE is not specified, the device's default power up value is used.

- Get the number of elements in the gain queue.

Message "?AISCAN:RANGE"

Response "AISCAN:RANGE=*value*"

value The number of elements in the gain queue.

- Add the range value as the next element in the gain queue, or set the range value for a specified channel (depending on the queue setting.

Message "AISCAN:RANGE{*ch*}=*value*"

Response "AISCAN:RANGE{*qcnt*/*ch*}" (when QUEUE is enabled)
"AISCAN:RANGE{*ch*}" (when QUEUE is disabled.)

The response behavior is dependent on the QUEUE setting:

- When QUEUE is enabled, adds the range value as the next element in the gain queue for the specified channel.
- When QUEUE is disabled, sets the range value for the specified channel.

qcnt the element's position in the gain queue. This number increments by 1 for each successive message sent.

ch The channel number.

value The range value (see values listed above)

Example "AISCAN:RANGE{2}=BIP20V"

- Set a specified element in the queue to a specified range (*value*) and channel (*ch*)

Message "AISCAN:RANGE{*element*/*ch*}=*value*"

Response "AISCAN:RANGE{*element*/*ch*}"

element The element's position in the gain queue.

ch The channel number.

value The range value (see values listed above).

Example "AISCAN:RANGE{0/1}=BIP20V"

Note If element is greater than the size of the queue, the size of the queue is expanded to *element + 1*.

- Get the range value for a specified element or channel.

Message "?AISCAN:RANGE{*x*}"

Response "AISCAN:RANGE{*element*/*ch*}=*value*" when QUEUE is enabled, or "AISCAN:RANGE{*ch*}=*value*" when QUEUE is disabled.

- When QUEUE is disabled, *x* denotes the channel for which the range is returned.
- When QUEUE is enabled, *x* denotes the element in the queue for which the range is returned.

value The range value (see values listed above)

ch The channel number.

element The element's position in the gain queue.

QUEUE

- Enable or disable the gain queue.

Message "AISCAN:QUEUE=*value*"

Response "AISCAN:QUEUE"

value ENABLE, DISABLE, or RESET

Example "AISCAN:QUEUE=ENABLE"

Note RESET resets the queue count to 0, and disables the gain queue.

- Read whether the gain queue is used in the scanning operation.

Message "?AISCAN:QUEUE"

Response "AISCAN:QUEUE=*value*"

value ENABLE, DISABLE

START

- Start an analog input scan.

Message "AISCAN:START"

Response "AISCAN:STATUS=RUNNING"

STOP

- Stop an analog input scan.

Message "AISCAN:STOP"

Response "AISCAN:STATUS=IDLE"

STATUS

- Get the status of the AISCAN operation.

Message "?AISCAN:STATUS"

Response "AISCAN:STATUS=*value*"

value IDLE, RUNNING, or OVERRUN

COUNT

- Get the number of samples per channel that have been acquired by the AISCAN operation.

Message "?AISCAN:COUNT"

Response "AISCAN:COUNT=*value*"

value The number of samples per channel acquired.

Example "AISCAN:COUNT=64"

Note This message is processed by the DAQFlex Software library, and is not sent to the device.

CAL

- Enable or disable calibration of the A/D data.

Message "AISCAN:CAL=*value*"

Response "AISCAN:CAL"

value ENABLE, DISABLE

Example "AISCAN:CAL=ENABLE"

Note This message is processed by the DAQFlex Software library, and is not sent to the device.

- get a value indicating whether calibration coefficients will be applied to the raw A/D data.

Message "?AISCAN:CAL"

Response "AISCAN:CAL=*value*"

value ENABLE, DISABLE

Example "?AISCAN:CAL"

Note This message is processed by the DAQFlex Software library, and is not sent to the device.

SCALE

- Enable or disable scaling of the A/D data.

Message "AISCAN:SCALE=*value*"

Response "AISCAN:SCALE"

value ENABLE, DISABLE

Example "AISCAN:SCALE=ENABLE"

Note This message is processed by the DAQFlex Software library, and is not sent to the device.

- Get a value indicating whether scaling will be applied to the A/D data.

Message "?AISCAN:SCALE"

Response "AISCAN:SCALE=*value*"

value ENABLE, DISABLE

Note This message is processed by the DAQFlex Software library, and is not sent to the device.

Working with the CAL and SCALE properties

The ENABLE/DISABLE setting of the CAL and SCALE properties affect the kind of data that is returned:

- CAL=DISABLE, SCALE=DISABLE

  If CAL and SCALE are both disabled, the data returned will be raw A/D integer values within the range of 0 to $2^{resolution}$ -1 of the device. If the calibration factors are stored on the device and applied to the data by the application software, the data range may be limited to well within these values.

- CAL=ENABLE, SCALE=DISABLE

  When CAL is enabled and SCALE is disabled, the format of the analog data returned will depend on the type of calibration implemented. If the calibration factors are stored on the device and applied to the data by the application software, the data will be floating point values, not integer values, and may exceed the theoretical limits and include negative values and values above $2^{resolution}$ -1.

- SCALE=ENABLE

  When SCALE is enabled, scaled floating point values are returned. The limits of the data will depend on the implementation of calibration, as described above. Data range limits may be a small percentage less than or greater than the full scale range selected for devices for which the calibration factors are stored on the device and applied to the data by the application software.

AITRIG

Sets and gets analog input trigger property values.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

TYPE, REARM

TYPE

- Set the edge trigger type.

Message	"AITRIG:TYPE=*value*"	
Response	"AITRIG:TYPE"	
	value	EDGE/RISING, EDGE/FALLING
Example	"AITRIG:TYPE=EDGE/RISING"	

- Get the edge trigger type.

Message	"?AITRIG:TYPE"	
Response	"AITRIG:TYPE=*value*"	
	value	EDGE/RISING, EDGE/FALLING

REARM

- Set the state of the retrigger mode.

Message	"AITRIG:REARM=*value*"	
Response	"AITRIG:REARM"	
	value	ENABLE, DISABLE
Example		

- Get the state of the retrigger mode.

Message	"?AITRIG:REARM"	
Response	"AITRIG:REARM=*value*"	
	value	ENABLE, DISABLE

Note

- When running the *DAQFlex test application*, AITRIG messages are listed on the AISCAN tab.

AO

Sets and gets property values for analog output channels.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

VALUE, SCALE

- Get the number of analog output channels on the device.

Message	"?AO"	
Response	"AO=*value*"	
	ch	The channel number.
	value	The number of D/A channels on the device

VALUE

- Set the value of a D/A channel.

Message	"AO{*ch*}:VALUE=*value*"	
Response	"AO{*ch*}:VALUE"	
	value	The D/A channel number.
Example	"AO{0}:VALUE=4095"	

SCALE

- Enable or disable scaling of all D/A channels.

Message	"AO:SCALE=*value*"	
Response	"AO:SCALE"	
	value	ENABLE, DISABLE
Example	"AO:SCALE=ENABLE"	
Note	This message is processed by the DAQFlex Software library, and is not sent to the device.	

- Get a value indicating whether scaling will be applied to the D/A channels.

Message	"?AO:SCALE"	
Response	"AO:SCALE=*value*"	
	ch	The channel number.
	value	ENABLE, DISABLE
Note	This message is processed by the DAQFlex Software library, and is not sent to the device.	

AOSCAN

Sets and gets property values when scanning analog output channels.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

RATE, SAMPLES, HIGHCHAN, LOWCHAN, RANGE, TRIG, START, STOP, STATUS, SCALE

RATE

- Set the D/A sampling rate in Hz for each channel.

Message	"AOSCAN:RATE=*value*"	
Response	"AOSCAN:RATE"	
	value	>0 (float)
Example	"AOSCAN:RATE=1000"	

- Get the D/A sampling rate in Hz for each channel.

Message	"?AOSCAN:RATE"	
Response	"?AOSCAN:RATE=*value*"	
	value	1 to the maximum rate of the device
Example	"?AOSCAN:RATE"	

SAMPLES

- Set the number of samples/channel to output.

Message	"AOSCAN:SAMPLES=*value*"	
Response	"AOSCAN:SAMPLES"	
	value	0 to N
Example	"AOSCAN{0}:SAMPLES=1000"	
Note	Set *value* to 0 for a continuous scan.	

- Get the number of samples/channel to output.

Message	"?AOSCAN:SAMPLES"	
Response	"AOSCAN:SAMPLES=*value*"	
	value	A numeric value.
Example	"?AOSCAN:SAMPLES"	

HIGHCHAN

- Set the last channel to include in the hardware-paced scan operation.

Message	"AOSCAN:HIGHCHAN=*value*"	
Response	"AOSCAN:HIGHCHAN"	
	value	Channel number
Example	"AOSCAN:HIGHCHAN=3"	

- Get the last channel to include in the hardware-paced scan operation.

Message	"?AOSCAN:HIGHCHAN"	
Response	"AOSCAN:HIGHCHAN=*value*"	
	value	Channel number
Example	"?AOSCAN:HIGHCHAN"	

LOWCHAN

- Set the first channel to include in the hardware-paced scan operation.

Message	"AOSCAN:LOWCHAN=*value*"	
Response	"AOSCAN:LOWCHAN"	
	value	Channel number
Example	"AOSCAN:LOWCHAN=0"	

- Get the first channel to include in the hardware-paced scan operation.

Message	"?AOSCAN:LOWCHAN"	
Response	"AOSCAN:LOWCHAN=*value*"	
	value	Channel number
Example	"?AOSCAN:LOWCHAN"	

RANGE

- Get the analog output range.

Message	"?AOSCAN:RANGE{*ch*}"	
Response	"AOSCAN:RANGE{*ch*}=*value*"	
	ch	Channel number
	value	UNI4V
Example	"AOSCAN:RANGE{0}=UNI4V"	

TRIG

- Enable or disable the trigger.

Message	"AOSCAN:TRIG=*value*"	
Response	"AOSCAN:TRIG"	
	value	ENABLE, DISABLE
Example	Example: "AOSCAN:TRIG=ENABLE"	
Note	If not set, TRIG is set to DISABLE by default.	

- Get the trigger status.

Message	"?AOSCAN:TRIG"	
Response	"AOSCAN:TRIG=*value*"	
	value	ENABLE, DISABLE
Example	"?AOSCAN:TRIG"	

START

- Start an analog output scan.

Message	"AOSCAN:START"
Response	"AOSCAN:STATUS=RUNNING"

STOP

- Stop an analog output scan.

Message	"AOSCAN:STOP"
Response	"AOSCAN:STATUS=IDLE"

STATUS

- Get the status of the AOSCAN operation.

Message "?AOSCAN:STATUS"

Response "AOSCAN:STATUS=*value*"

value IDLE, RUNNING, or UNDERRUN

Example "AOSCAN:STATUS=RUNNING"

SCALE

- Enable or disable scaling of the D/A data.

Message "AOSCAN:SCALE=*value*"

Response "AOSCAN:SCALE"

value ENABLE, DISABLE

Example "AOSCAN:SCALE=ENABLE"

Note This message is processed by the MC7000 DAQ Software library, and is not sent to the device.

- Get a value indicating whether scaling will be applied to the D/A data.

Message "?AOSCAN:SCALE"

Response "AOSCAN:SCALE=*value*"

value ENABLE, DISABLE

Note This message is processed by the MC7000 DAQ Software library, and is not sent to the device.

AOTRIG

Sets and gets analog output trigger property values.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

TYPE

TYPE

- Set the trigger edge.

Message	"AOTRIG:TYPE=*value*"	
Response	"AOTRIG:TYPE"	
	value	EDGE/RISING, EDGE/FALLING
Example	"AOTRIG:TYPE=EDGE/RISING"	

- Get the trigger edge.

Message	"?AOTRIG:TYPE"	
Response	"AOTRIG:TYPE=*value*"	
	value	EDGE/RISING, EDGE/FALLING

CTR

Sets and gets property values for counter channels.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

VALUE, START, STOP

- Get the number of counter channels on a device.

Message	"?CTR"	
Response	"CTR=*value*"	
	value	The number of counter channels on a device.

VALUE

- Load the specified counter channel with a value.

Message	"CTR{ch}:VALUE=*value*"	
Response	"CTR{ch}:VALUE"	
	ch	The number of the counter channel.
	value	The value to load onto the counter channel.
Example	"CTR{0}:VALUE=0"	

- Get the value of the specified counter channel.

Message	"?CTR{ch}:VALUE"	
Response	"CTR{ch}:VALUE=*value*"	
	ch	The number of the counter channel.
	value	The value of the counter channel.
Example	"?CTR{0}:VALUE"	

START

- Start the specified counter channel.

Message	"CTR{ch}:START"	
Response	"CTR{ch}:START"	
	ch	The number of the counter channel.
Example	"CTR{0}:START"	

STOP

- Stop the specified counter channel.

Message	"CTR{ch}:STOP"	
Response	"CTR{ch}:STOP"	
	ch	The number of the counter channel.
Example	"CTR{0}:STOP"	

DEV

Sets and gets device property values.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

MFGSER, FWV, FLASHLED, ID, MFGCAL, RESET, DATATYPE

MFGSER

- Get the manufacturer's device serial number.

Message	"?DEV:MFGSER"	
Response	"DEV:MFGSER=*value*"	
	value	The serial number of the device.
Example	"DEV:MFGSER=00000001"	

FWV

- Get the firmware version.

Message	"?DEV:FWV"	
Response	"DEV:FWV=*value*"	
	value	The firmware version.
Example	"DEV:FWV=01.02" or "DEV:FWV=01.01.00f00"	

FLASHLED

- Flash the device LED.

Message	"DEV:FLASHLED/{n}"	
Response	"DEV:FLASHLED"	
	n	A number indicating how many times to flash the device LED.
Example	"DEV:FLASHLED/5"	

ID

- Set the device ID.

Message	"DEV:ID=*value*"	
Response	"DEV:ID"	.
	value	The ID set for the device. Set to MYDEVICE by default.
Example	"DEV:ID=MYDEVICE"	
Note	*value* can be up to 57 characters.	

- Get the device ID.

Message	"?DEV:ID"	
Response	"DEV:ID=*value*"	.
	value	The ID set for the device.

MFGCAL

- Get the date and time in which the device was last calibrated.

Message "?DEV:MFGCAL"

Response "DEV:MFGCAL=*value*"

value The calibration date and time.

Example "DEV:MFGCAL=2009-03-14 13:56:27"

- Get the year in which the device was last calibrated.

Message "?DEV:MFGCAL{YEAR}"

Response "DEV:MFGCAL{YEAR}=*value*"

YEAR The calibration year.

Example "DEV:MFGCAL{YEAR}=2009"

- Get the month in which the device was last calibrated.

Message "?DEV:MFGCAL{MONTH}"

Response "DEV:MFGCAL{MONTH}=*value*"

MONTH The calibration month.

Example "DEV:MFGCAL{MONTH}=03"

- Get the day in which the device was last calibrated.

Message "?DEV:MFGCAL{DAY}"

Response "DEV:MFGCAL{DAY}=*value*"

DAY The calibration day.

Example "DEV:MFGCAL{DAY}=14"

- Get the hour in which the device was last calibrated.

Message "?DEV:MFGCAL{HOUR}"

Response "DEV:MFGCAL{HOUR}=*value*"

HOUR The calibration hour.

Example "DEV:MFGCAL{HOUR}=15"

- Get the minute in which the device was last calibrated.

Message "?DEV:MFGCAL{MINUTE}"

Response "DEV:MFGCAL{MINUTE}=*value*"

MINUTE The calibration minute.

Example "DEV:MFGCAL{MINUTE}=56"

- Get the second in which the device was last calibrated.

Message "?DEV:MFGCAL{SECOND}"

Response "DEV:MFGCAL{SECOND}=*value*"

SECOND The calibration second.

Example "DEV:MFGCAL{SECOND}=26"

RESET

- Reset the device or the default parameters.

Message "DEV:RESET/*value*"

Response

- "NO COMPONENT SET" when *value* is set to SYSTEM.

- "DEV:RESET" when *value* is set to DEFAULT.

value SYSTEM, DEFAULT

Example "DEV:RESET/DEFAULT"

Note
- *SYSTEM* resets the USB interface to the device (not recommended for use through the API).
- *DEFAULT* resets all device parameters to the default value.

DATATYPE

- Set whether the datatype is included in raw data returns.

Message "DEV:DATATYPE=*value*"

Response "DEV:DATATYPE"

value ENABLE, DISABLE

Example "DEV:DATATYPE=ENABLE"

Note For more information on DATATYPE, refer to the *Message-based Firmware Specification* help.

DIO

Sets and gets property values for digital I/O channels.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

DIR, VALUE

- Get the number of digital ports on a device.

Message	"?DIO"	
Response	"DIO=*value*"	
	value	The number of digital ports.

- Get the number of bits on a port.

Message	"?DIO{bit}"	
Response	"DIO{bit}=*value*"	
	value	The number of bits on the port.
Example	"?DIO{0}"	

DIR

- Set the direction of a port.

Message	"DIO{port}:DIR=*value*"	
Response	"DIO{port}:DIR"	
	port	The port number.
	value	IN, OUT
Example	"DIO{0}:DIR=IN"	
Note	If DIR is not specified, the devise's default power up value is used.	

- Get the direction of a port.

Message	"?DIO{port}:DIR"	
Response	"DIO{port}:DIR=*value*"	
	port	The port number.
	value	IN, OUT, or a number between 0 and 2^n -1, where *n* is the number of bits in the port.
Note	■ If bits are not individually configurable, the value returned is either "IN" or "OUT." ■ If each bit is individually configurable, *value* is a bit mask, in which *1* indicates that the bit is configured for input, and *0* indicates that the bit is configured for output.	

- Set the direction of a bit.

Message	"DIO{port/bit}:DIR=*value*"	
Response	"DIO{port/bit}:DIR"	
	port	The port number.
	bit	The bit number.
	value	IN, OUT
Example	"DIO{0/1}:DIR=IN"	
Note	If DIR is not specified, the device's default power up value is used.	

- Get the direction of a bit.

Message "?DIO{port/bit}:DIR"

Response "DIO{port/bit}:DIR=*value*"

port The port number.

bit The bit number.

value IN, OUT

Example "?DIO{0/1}:DIR"

VALUE

- Set the value of a port.

Message "DIO{port}:VALUE=*value*"

Response "DIO{port}:VALUE"

port The port number.

value The value of the port.

Example "DIO{0}:VALUE=128"

- Get the value of a port.

Message "?DIO{port}:VALUE"

Response "DIO{port}:VALUE=*value*"

port The port number.

value The value of the port.

Example "?DIO{0}:VALUE"

- Set the value of a bit.

Message "DIO{port/bit}:VALUE=*value*"

Response "DIO{port/bit}:VALUE"

port The port number.

bit The bit number.

value The value of the bit (0 or 1).

Example "DIO{0/1}:VALUE=1"

- Get the value of a bit.

Message "?DIO{port/bit}:VALUE"

Response "DIO{port/bit}:VALUE=*value*"

port The port number.

bit The bit number.

value The value of the bit (0 or 1).

Example "?DIO{0/1}:VALUE"

TMR

Sets and gets property values for timer output channels.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

PULSE, DUTYCYCLE, START, STOP

PULSE

- Get the number of timer output channels on the device.

Message	"?TMR"	
Response	"TMR=*value*"	
	value	The number of timer output channels.

- Set the value in Hz of the pulse frequency for the specified channel.

Message	"TMR{*ch*}:PULSE=*value*"	
Response	"TMR{*ch*}:PULSE"	
	ch	The channel number.
	value	The pulse frequency in Hz.
Example	"TMR{0}:PULSE=1000"	

- Get the value in Hz of the pulse frequency for the specified channel.

Message	"?TMR{*ch*}:PULSE"	
Response	"TMR{*ch*}:PULSE=*value*"	
	ch	The channel number.
	value	The pulse frequency in Hz.
Example	"?TMR{0}:PULSE"	

DUTYCYCLE

- Set the value in percent of the duty cycle for the specified channel.

Message	"TMR{*ch*}:DUTYCYCLE=*value*"	
Response	"TMR{*ch*}:DUTYCYCLE"	
	ch	The channel number.
	value	The duty cycle in percent (%).
Example	"TMR{0}:DUTYCYCLE=1000"	

- Get the value of the duty cycle in percent for the specified channel.

Message	"?TMR{*ch*}:DUTYCYCLE"	
Response	"TMR{*ch*}:DUTYCYCLE=*value*"	
	ch	The channel number.
	value	The duty cycle in percent (%).
Example	"?TMR{0}:DUTYCYCLE"	

START

- Start the timer output.

Message	"TMR:START"
Response	"TMR:START"

STOP

- Stop the timer output.

Message	"TMR:STOP"
Response	"TMR:STOP"

Device reflection messages

Device reflection messages get information about the capabilities of a device, such as the maximum scan rate or support for an external clock. Device reflection messages always start with the **@** character.

Device features are stored on the device in EEPROM.

Click on a component below for the string messages, device responses, and property values supported by the component.

Components for retrieving device information	Description
AI	Gets the analog input properties of a device.
AISCAN	Gets the analog input scanning properties of a device.
AITRIG	Gets the analog input triggering properties of a device.
AO	Gets the analog output properties of a device.
AOSCAN	Gets the analog output scanning properties of a device.
DIO	Gets the digital I/O properties of a device.
CTR	Gets the counter input properties of a device.
TMR	Gets the timer output properties of a device.

AI

Gets the analog input properties of a device.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

CHANNELS, MAXCOUNT, MAXRATE, RANGES, CHMODE, SENSORS, SENSORCONFIG, TCTYPES, CJC, FACCAL, FIELDCAL

CHANNELS

- Get the number of analog input channels on a device.

Message	"@AI:CHANNELS"	
Response	"AI:CHANNELS=*value*"	
	value	The number of A/D channels on a device.
Example	"AI:CHANNELS=8"	

MAXCOUNT

- Get the maximum count of the device's A/D converter.

Message	"@AI:MAXCOUNT"	
Response	"AI:MAXCOUNT=*value*"	
	value	The maximum count of the A/D converter.
Example	"AI:MAXCOUNT=65535"	

MAXRATE

- Get the maximum rate for software paced analog input operations.

Message	"@AI:MAXRATE"	
Response	"AI:MAXRATE=*value*"	
	value	The maximum sampling rate of the device.
Example	"AI:MAXRATE=100"	
Note	The maximum rate is based on the device's ability to perform single-point I/O.	

RANGES

- Get the analog input ranges that are supported by the specified channel.

Message	"@AI{ch}:RANGES"	
Response	"AI{ch}:RANGES=<*implementation*><*value*>"	
	ch	Channel number.
	implementation	FIXED%, PROG% (programmable), or HWSEL% (hardware selectable)
	value	BIP, UNI, or TO (current range)
Example	"AI{0}:RANGES=PROG%BIP10V,BIP5V"	
Note	On some devices, the values returned may be dependent on chanel configuration settings.	

CHMODE

- Get the analog input channel modes that are supported by the specified channel.

Message	"@AI{ch}:CHMODE"	
Response	"AI{ch}:CHMODE=<*implementation*><*value*>"	
	ch	Channel number.
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)

	value	SE or DIFF
Example	"AI{0}:CHMODE=PROG/SE,DIFF"	

INPUTS

- Get the analog input signal types that are supported by the specified channel.

Message	"@AI{ch}:INPUTS"	
Response	"AI{ch}:INPUTS=<*implementation*><*value*>"	
	ch	Channel number.
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)
	value	VOLTS, TEMP, CUR (current), or RES (resistance)
Example	"AI{0}:INPUTS=PROG%VOLTS,TEMP"	

SENSORS

- Get the analog input sensor types that are supported by the specified channel.

Message	"@AI{ch}:SENSORS"	
Response	"AI{ch}:SENSORS=<*implementation*><*value*>"	
	ch	Channel number.
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)
	value	TC, RTD, THERM (thermistor), or SEMI (semiconductor)
Example	"AI{0}:SENSORS=FIXED%TC"	

SENSORCONFIG

- Get the analog sensor configurations that are supported by the specified channel.

Message	"@AI{ch}:SENSORCONFIG"	
Response	"AI{ch}:SENSORCONFIG=<*implementation*><*value*>"	
	ch	Channel number.
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)
	value	2WIRE, 3WIRE, 4WIRE, FULLBRG, HALFBRG, QTRBRG
Example	"AI{0}:SENSORCONFIG=PROG%2WIRE,3WIRE,4WIRE"	

TCTYPES

- Get the thermocouple sensor types that are supported by the specified channel.

Message	"@AI{ch}:TCTYPES"	
Response	"AI{ch}:TCTYPES=<*implementation*><*value*>"	
	ch	Channel number.
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)
	value	B, E, J, K, N, R, S, or T
Example	"AI{0}:TCTYPES=PROG%B,E,J,K,N,R,S,T"	

CJC

- Get the CJC channel number associated with an analog input channel.

Message	"@AI{ch}:CJC"	
Response	"AI{ch}:CJC=<*implementation*><*value*>"	
	ch	Channel number.
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)

value Channel number

Example "AI{0}:CJC=FIXED%0"

FACCAL

- Get a value indicating if the device supports factory calibration for analog inputs.

Message "@AI:FACCAL"

Response "AI:FACCAL=*value*"

value YES or NO

Example "AI:FACCAL=YES"

FIELDCAL

- Get a value indicating if the device supports field calibration for analog inputs.

Message "@AI:FIELDCAL"

Response "AI:FIELDCAL=*value*"

value MANUAL, AUTO, or NOT_SUPPORTED

Example "AI:FIELDCAL=AUTO"

AISCAN

Get the analog input scan properties of a device.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

MAXSCANTHRUPUT, MINSCANRATE, MAXSCANRATE, MAXBURSTTHRUPUT, MINBIRSTRATE, MAXBIURSTRATE, CHQUEUE, GAINQUEUE, XFRMODES, EXTTRIG

MAXSCANTHRUPUT

- Get the maximum analog input throughput.

Message	"@AISCAN:MAXSCANTHRUPUT"	
Response	"AISCAN:MAXSCANTHRUPUT=*value*"	
	value	The maximum throughput rate.
Example	"AISCAN:MAXSCANTHRUPUT=200000"	

MINSCANRATE

- Get the minimum hardware-paced input scan rate.

Message	"@AISCAN:MINSCANRATE"	
Response	"AISCAN:MINSCANRATE=*value*"	
	value	The minimum input scan rate.
Example	"AISCAN:MINSCANRATE=0.569"	

MAXSCANRATE

- Get the maximum hardware-paced input scan rate.

Message	"@AISCAN:MAXSCANRATE"	
Response	"AISCAN:MAXSCANRATE=*value*"	
	value	The maximum input scan rate.
Example	"AISCAN:MAXSCANRATE=1000"	

MAXBURSTTHRUPUT

- Get the maximum analog input throughput for BURSTIO mode operations.

Message	"@AISCAN:MAXBURSTTHRUPUT"	
Response	"AISCAN:MAXBURSTTHRUPUT=*value*"	
	value	The maximum analog input throughput.
Example	"AISCAN:MAXBURSTTHRUPUT=2000"	

MINBURSTRATE

- Get the minimum hardware-paced input scan rate for BURSTIO mode operations.

Message	"@AISCAN:MINBURSTRATE"	
Response	"AISCAN:MINBURSTRATE=*value*"	
	value	The minimum scan rate for BURSTIO mode operations.
Example	"AISCAN:MINBURSTRATE=20"	

MAXBURSTRATE

- Get the maximum hardware-paced input scan rate for BURSTIO mode operations.

Message	"@AISCAN:MAXBURSTRATE"

Response "AISCAN:MAXBURSTRATE=*value*"

value The maximum scan rate for BURSTIO mode operations.

Example "AISCAN:MAXBURSTRATE=200000"

CHQUEUE

- Get the number of channels in the device's analog input channel queue.

Message "@AISCAN:CHQUEUE"

Response "AISCAN:CHQUEUE=*value*"

value The number of channels in the channel queue.

Example "AISCAN:CHQUEUE=8"

GAINQUEUE

- Get the number of channels in the device's analog input gain queue.

Message "@AISCAN:GAINQUEUE"

Response "AISCAN:GAINQUEUE=*value*"

value The number of channels in the gain queue.

Example "AISCAN:GAINQUEUE=8"

XFRMODES

- Get the input transfer modes supported by a device.

Message "@AISCAN:XFRMODES"

Response "AISCAN:XFRMODES=<*implementation*><*value*>"

implementation FIXED%, PROG% (programmable), HWSEL% (hardware selectable)

value BLOCKIO, SINGLEIO, BURSTAD, BURSTFIFO, or REARM

Example "AISCAN:XFRMODES=PROG%BLOCKIO,SINGLEIO,BURSTFIFO"

PACERSRC

- Get a value indicating which A/D pacing sources are supported by the device.

Message "@AISCAN:PACERSRC"

Response "AISCAN:PACERSRC=<*implementation*><*value*>"

implementation FIXED%, PROG% (programmable), HWSEL% (hardware selectable)

value INT (internal), EXT_MSTR (external master), EXT_SLV (external slave), EXT_GSLV (external gslave)

Example "AISCAN:PACERSRC=PROG%INT,EXT_SLV"

EXTTRIG

- Get a value indicating whether the device supports external triggering of analog input channels.

Message "@AISCAN:EXTTRIG"

Response "AISCAN:EXTTRIG=*value*"

value SUPPORTED, NOT_SUPPORTED

Example "AISCAN:EXTTRIG=SUPPORTED"

FIFOSIZE

- Get the size in bytes of the device's FIFO.

Message "@AISCAN:FIFOSIZE"

Response "AISCAN:FIFOSIZE=*value*"

	value	The size, in bytes, of the device's FIFO.

Example "AISCAN:FIFOSIZE=4096"

AITRIG

Get the analog input trigger properties of a device.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

SOURCES, TYPES, RANGES, MAXCOUNT

SOURCES

- Set the edge trigger type.

Message	"@AITRIG:SOURCES"	
Response	"AITRIG:SOURCES=<*implementation*><*value*>"	
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)
	value	HW/<hw trigger>
		SWSTART/<sw start trigger>
		SWSTOP/<sw stop trigger>
	hw trigger	DIG, ANLG
	sw start trigger	<component><ch>
	sw stop trigger	<component><ch>
	component	AI, DIO, CTR
	ch	Channel number
Example	"AITRIG:SOURCES=PROG%HW/DIG,HW/ANLG"	

TYPES

- Get the types of trigger sensing that are supported by the device.

Message	"@AITRIG:TYPES"	
Response	"AITRIG:TYPES=<*implementation*><*value*>"	
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)
	value	<type>/<condition>
	type	EDGE, LEVEL, GATE
	condition	RISING, FALLING, HIGH, LOW, ABOVE, BELOW, OUTWINDOW
Example	"AITRIG:TYPES=PROG%LEVEL/HIGH,LEVEL/LOW"	

RANGES

- Get the supported ranges for a device's analog input trigger circuit.

Message	"@AITRIG:RANGES"	
Response	"AITRIG:RANGES=<*implementation*><*value*>"	
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)
	value	BIP, UNI, or TO (current range)
Example	"AITRIG:RANGES=PROG%BIP10V,BIP5V"	
Note	On some devices, the values returned may be dependent on chanel configuration settings.	

MAXCOUNT

- Get the maximum count of the device's A/D trigger circuit.

Message	"@AITRIG:MAXCOUNT"

Response "AITRIG:MAXCOUNT=*value*"

value The maximum count of the trigger circuit.

Example "AITRIG:MAXCOUNT=4095"

AO

Gets the analog output properties of a device.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

CHANNELS, MAXCOUNT, MAXRATE, OUTPUTS, FACCAL, FIELDCAL

CHANNELS

- Get the number of analog output channels on a device.

Message	"@AO:CHANNELS"	
Response	"AO:CHANNELS=*value*"	
	value	The number of D/A channels on a device.
Example	"AO:CHANNELS=4"	

MAXCOUNT

- Get the maximum count of the device's D/A converter.

Message	"@AO:MAXCOUNT"	
Response	"AO:MAXCOUNT=*value*"	
	value	The maximum count of the D/A converter.
Example	"AO:MAXCOUNT=65535"	

MAXRATE

- Get the Returns the maximum rate for software paced analog output operations.

Message	"@AO:MAXRATE"	
Response	"AO:MAXRATE=*value*"	
	value	The maximum output rate of the device.
Example	"AO:MAXRATE=100"	
Note	The maximum rate is based on the device's ability to perform single-point I/O.	

OUTPUTS

- Get the analog input signal types that are supported by the specified channel.

Message	"@AO{ch}:OUTPUTS"	
Response	"AO{ch}:OUTPUTS=<*implementation*><*value*>"	
	ch	Channel number.
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)
	value	VOLTS, CUR
Example	"AO{0}:OUTPUTS=PROG%VOLTS,CUR"	

FACCAL

- Get a value indicating if the device supports factory calibration for analog outputs.

Returns

Message	"@AO:FACCAL"	
Response	"AO:FACCAL=*value*"	
	value	YES or NO
Example	"AO:FACCAL=YES"	

FIELDCAL

- Get a value indicating if the device supports field calibration for analog outputs.

Message	"@AO:FIELDCAL"	
Response	"AO:FIELDCAL=*value*"	
	value	MANUAL, AUTO, or NOT_SUPPORTED
Example	"AO:FIELDCAL=AUTO"	

AOSCAN

Gets the analog output scan properties of a device.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

MINSCANRATE, MAXSCANRATE, PACERSRC, EXTTRIG, ADCLKTRIG, SIMUL

MINSCANRATE

- Get the minimum hardware-paced output scan rate.

Message	"@AOSCAN:MINSCANRATE"	
Response	"AOSCAN:MINSCANRATE=*value*"	
	value	The minimum output scan rate.
Example	"AOSCAN:MINSCANRATE=1"	

MAXSCANRATE

- Get the maximum hardware-paced output scan rate.

Message	"@AOSCAN:MAXSCANRATE"	
Response	"AOSCAN:MAXSCANRATE=*value*"	
	value	The maximum output scan rate.
Example	"AOSCAN:MAXSCANRATE=1000"	

PACERSRC

- Get a value indicating which D/A pacing sources are supported by the device.

Message	"@AOSCAN:PACERSRC"	
Response	"AOSCAN:PACERSRC=<*implementation*><*value*>"	
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)
	value	INT (internal), EXT_MSTR (external master), EXT_SLV (external slave), EXT_GSLV (external gslave)
Example	"AOSCAN:PACERSRC=PROG%INT,EXT_SLV"	

EXTTRIG

- Get a value indicating whether analog output channels can be externally triggered.

Message	"@AOSCAN:EXTTRIG"	
Response	"AOSCAN:EXTTRIG=*value*"	
	value	SUPPORTED, NOT_SUPPORTED
Example	"AOSCAN:EXTTRIG=SUPPORTED"	

ADCLKTRIG

- Get a value indicating whether analog output channels can be triggered by the device's A/D clock.

Message	"@AOSCAN:ADCLKTRIG"	
Response	"AOSCAN:ADCLKTRIG=*value*"	
	value	SUPPORTED, NOT_SUPPORTED
Example	"AOSCAN:ADCLKTRIG=SUPPORTED"	

SIMUL

- Get a value indicating whether analog output channels can be updated simultaneously.

Message	"@AOSCAN:SIMUL"
Response	"AOSCAN:SIMUL=*value*"
	value SUPPORTED, NOT_SUPPORTED
Example	"AOSCAN:SIMUL=SUPPORTED"

CTR

Gets the counter channel properties of a device.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

CHANNELS, MAXCOUNT, TYPE, EDGE

CHANNELS

- Get the number of conter channels on a device.

Message	"@CTR:CHANNELS"	
Response	"CTR:CHANNELS=*value*"	
	value	The number of counter channels on a device.
Example	"CTR:CHANNELS=3"	

MAXCOUNT

- Get the maximum count of the specified counter.

Message	"@CTR{ch}:MAXCOUNT"	
Response	"CTR{ch}:MAXCOUNT=*value*"	
	ch	The number of the counter channel.
	value	The maximum count of the counter.
Example	"CTR{0}:MAXCOUNT=65535"	

TYPE

- Get the counter type.

Message	"@CTR{ch}:TYPE"	
Response	"CTR{ch}:TYPE=*value*"	
	ch	The number of the counter channel.
	value	TYPE1 (8254), TYPE2 (9513)
Example	"CTR{0}:TYPE=TYPE1"	

EDGE

- Get a value indicating whether a counter's edge detection is programmable.

Message	"@CTR{ch}:EDGE"	
Response	"CTR{ch}:EDGE=<*implementation*><*value*>"	
	ch	The number of the counter channel.
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)
	value	RISING, FALLING
Example	"CTR{0}:EDGE=PROG%RISING,FALLING"	

DIO

Gets the digital I/O properties of a device.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

CHANNELS, MAXCOUNT, CONFIG

CHANNELS

- Get the number of digital channels (ports) on a device.

Message	"@DIO:CHANNELS"	
Response	"DIO:CHANNELS=*value*"	
	value	The number of digital channels (ports) on a device.
Example	"DIO:CHANNELS=3"	

MAXCOUNT

- Get the maximum count of the specified port.

Message	"@DIO{ch}:MAXCOUNT"	
Response	"DIO{ch}:MAXCOUNT=*value*"	
	ch	The digital port number.
	value	The maximum count of the digital port.
Example	"DIO{0}:MAXCOUNT=65535"	

CONFIG

- Get the configuration options of the specified port.

Message	"@DIO{ch}:CONFIG"	
Response	"DIO{ch}:CONFIG=<*implementation*><*value*>"	
	ch	The digital port number.
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)
	value	BITIN, BITOUT, PORTIN, PORTOUT
Example	"DIO{0}:CONFIG=PROG%BITIN,BITOUT,PORTIN,PORTOUT"	

TMR

Gets the timer output properties of a device.

Refer to the device-specific information in the *Hardware Reference* section for the component properties and commands supported by each DAQ device.

Properties

CHANNELS, MAXCOUNT, CLKSRC, BASEFREQ

CHANNELS

- Get the number of timer outputs on a device.

Message	"@TMR:CHANNELS"	
Response	"TMR:CHANNELS=*value*"	
	value	The number of timer outputs on a device.
Example	"TMR:CHANNELS=2"	

MAXCOUNT

- Get the maximum count of a specified timer.

Message	"@TMR{ch}:MAXCOUNT"	
Response	"TMR{ch}:MAXCOUNT=*value*"	
	ch	The number of the timer channel.
	value	The maximum count of the timer.
Example	"TMR{0}:MAXCOUNT=65535"	

CLKSRC

- Get the clock sources that are available for a specified timer.

Message	"@TMR{ch}:CLKSRC"	
Response	"TMR:CLKSRC=<*implementation*><*value*>"	
	ch	The timer number.
	implementation	FIXED%, PROG% (programmable), HWSEL% (hardware selectable)
	value	INT, EXT
Example	"TMR{0}:CLKSRC=PROG%INT,EXT"	

BASEFREQ

- Get the frequency of the specified timer's internal base clock.

Message	"@TMR{ch}:BASEFREQ"	
Response	"TMR{ch}:BASEFREQ=*value*"	
	ch	The number of the timer channel.
	value	The frequency of the timer's internal clock.
Example	"TMR{0}:BASEFREQ=10000000"	

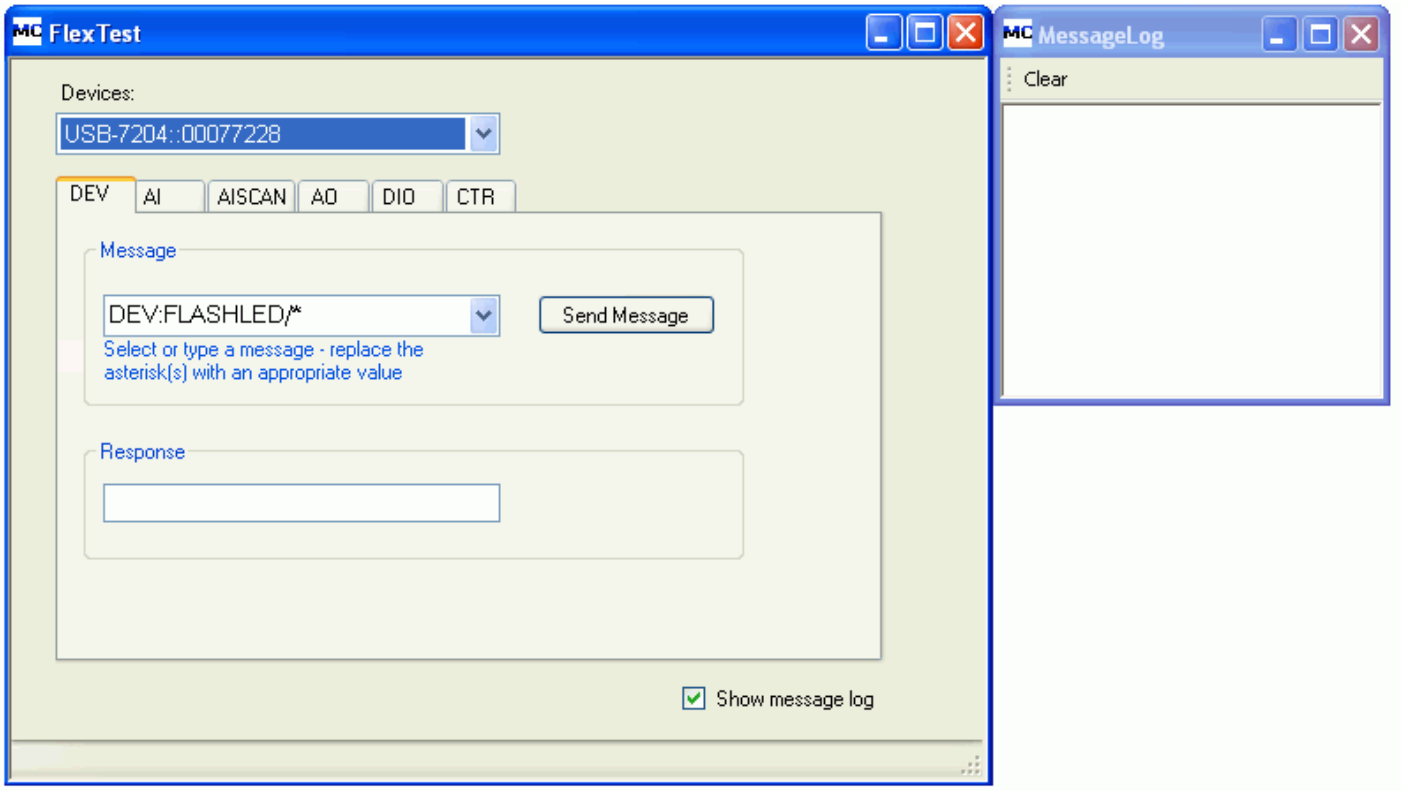
FlexTest application

FlexTest is an interactive C# console application that demonstrates how to communicate with a device using the DAQFlex communication protocol and software API. The source code is included.

- Windows 7 and Windows Vista, FlexTest is installed to `C:\Users\Public Documents\Measurement Computing\DAQFlex For Windows\FlexTest.exe`.
- On Windows XP, FlexTest is installed to `C:\Program Files\Measurement Computing\DAQFlex For Windows\FlexTest.exe`.
- On Windows CE, FlexTest is installed to `C:\Program Files\Measurement Computing\DAQFlex For Windows CE\FlexTest.exe`.
- On Mac® OS X, FlexTest is installed to the `/Applications` folder.
- On Linux®, FlexTest is extracted to `DAQFlex/FlexTest.exe`.

Note: Connect a device that supports the DAQFlex protocol before running FlexTest.

To run FlexTest, go to **Start »Programs » Measurement Computing » DAQFlex » FlexTest**. When run, the main **FlexTest** window and a **MessageLog** window open. An example is shown here:



The FlexTest window features the following controls:

- **Devices** drop-down list: displays the name and serial number of each connected DAQFlex-supported device.
- **Component** tabs: the DAQ components that are supported by the DAQFlex-supported device.
- **Message** field and drop down list: displays the text messages that can be sent to the device. The messages are specific to the component selected. An asterisk in the message indicates a variable whose property value must be entered.

  Refer to the DAQFlex Message Reference topic for more information about the API messages.
- **Send Message** button: click this button to send the selected message to the device.
- **Response** field: Displays the response to the message that is returned from the device.

  When a number is returned from the device, for example when reading the value of an analog input channel, options to display the value as either a text string or a numeric appear to the right of the Response field.

  When multiple values are returned, such as when scanning data, the values appear below the Response field.
- **Show message log** check box: When checked, the text of each message sent to the device appears in the **MessageLog** window.
- **Status area**: The lower left corner of the window displays the following information:
  - *Message status*: When a message is successfully sent to a device, "Success" appears in the status area. If a message cannot be sent, such as when a variable is either not set or is set to an unsupported value, an error message appears in the status area.
  - *Scan count*: during a scan operation, the status area updates with the number of samples that are read.

The **MessageLog** window lists each message that is sent to the device. The **Clear** menu option removes all messages from this window.

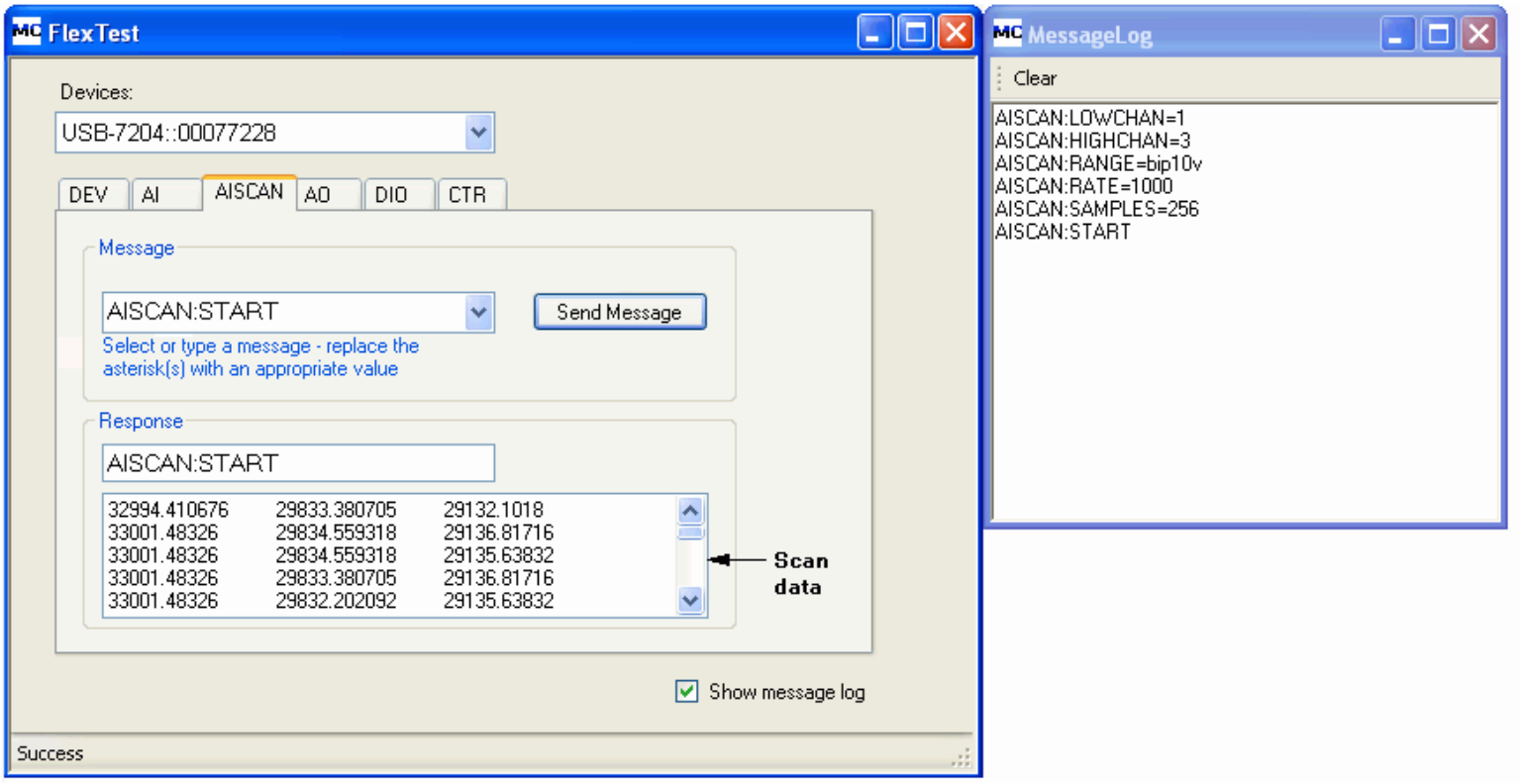
Using FlexTest

The procedure below demonstrates how to read and display multiple channel scan data using FlexTest.

For this exercise, you set the range of analog channels to scan (1 to 3), set the channel range to ±10 volts, the sample rate to 1000 Hz, and the number of samples to 256. After configuring the scan parameters, you run the operation and view the resulting scan data.
Do the following:

1. Connect a device that supports the DAQFlex protocol to your system and run FlexTest.
2. Click on the **AISCAN** tab.
3. Configure the scan parameters using the text strings in the **Message** drop down list:
   - Select **AISCAN:LOWCHAN=***. Highlight the asterisk and enter "1", then click **Send Message**.
   - Select **AISCAN:HIGHCHAN=***. Highlight the asterisk and enter "3", then click **Send Message**.
   - Select **AISCAN:RANGE=***. Highlight the asterisk and enter "BIP10V", then click **Send Message**. This field is *not* case-sensitive.
   - Select **AISCAN:RATE=***. Highlight the asterisk and enter "1000", then click **Send Message**.
   - Select **AISCAN:SAMPLES=***. Highlight the asterisk and enter "256", then click **Send Message**.
4. Start the scan operation: Select **AISCAN:START** and click **Send Message**.

The DAQFlex Test window displays the scan data, and the MessageLog window lists the messages sent to the device.



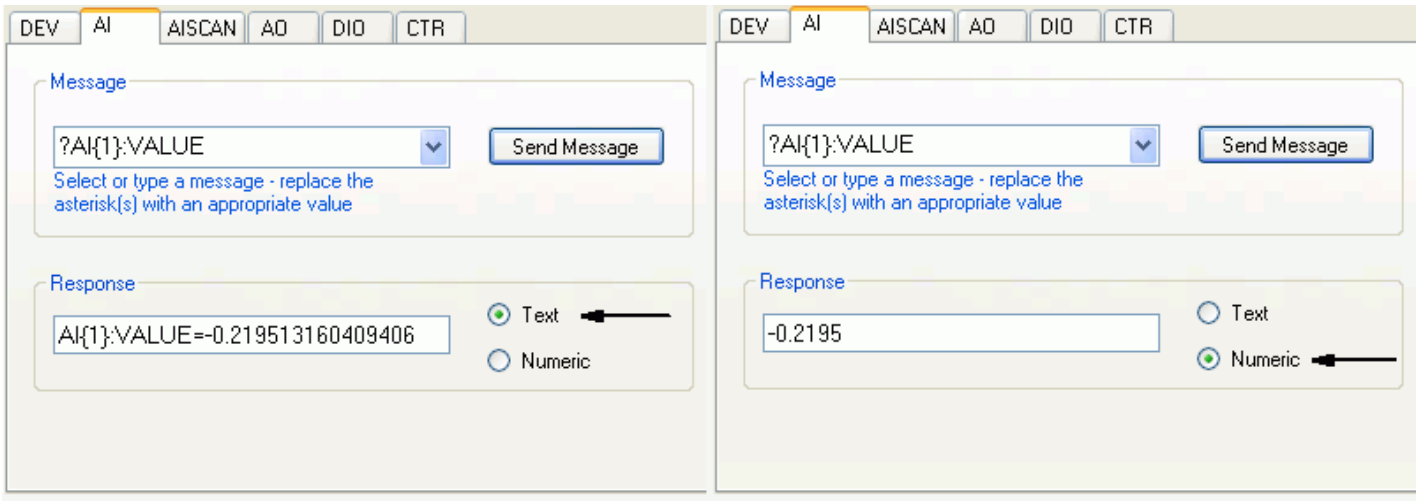
Notes

- String data entered into the Message field is not case sensitive.
- Messages that begin with "?" are query messages. Select a query to read a value.

Text and numeric data

Data is returned by default as a text string. If numeric data is supported for the returned data, **Text** and **Numeric** radio buttons appear next to the Response field.

- Select the **Text** radio button to display the returned value in a text string.
- Select the **Numeric** radio button to display the returned value as raw data.



Note: Click **Send Message** after changing the type of data to return.

API message reference

Refer to the API messages topic for information about the DAQFlex API messages.

C# and VB .NET example programs

Complete C# and VB example programs are installed with DAQFlex that demonstrate how to configure DAQFlex-supported devices and perform DAQ operations with the DAQFlex Software API.

- On Windows, the example programs are installed to CSharp and VB subfolders:

  Windows 7 and Windows Vista — `C:\Users\Public Documents\Measurement Computing\DAQFlex For Windows\Examples`.

  Windows XP — `C:\Program Files\Measurement Computing\DAQFlex For Windows\Examples`.

  Windows CE — `C:\Program Files\Measurement Computing\DAQFlex For Windows CE\Examples`.

- On Mac OS X, example programs are installed to `Users/Shared/Measurement Computing/DAQFlex/Examples`.
- On Linux, C# example programs are extracted to `DAQFlex/Examples/CSharp`.

Hardware Reference

Select your DAQFlex-supported device below for the components and programming messages supported by the device.

Note: You can use any of the device reflection messages to retrieve information about the device's capabilities.

- USB-2001-TC
- USB-7202
- USB-7204

USB-2001-TC

Use the components below to set or get device properties.

Component	Supported Property/Command	Set/Get	Supported Values
DEV	MFGSER	Get	Up to 8 numeric characters
	FWV	Get	Firmware version
	ID	Set/Get	Up to 56 characters
	MFGCAL	Get	yyyy-mm-dd HH:MM:SS
	MFGCAL{YEAR}	Get	Year as yyyy
	MFGCAL{MONTH}	Get	Month as mm
	MFGCAL{DAY}	Get	Day as dd
	MFGCAL{HOUR}	Get	0 - 59
	MFGCAL{MINUTE}	Get	0 - 59
	MFGCAL{SECOND}	Get	0
	FLASHLED		0 - 255
AI	RANGE{ch}	Set/Get	BIP73.125E-3V (±0.073125 volts) BIP146.25E-3V (±0.14625 volts)
	SENSOR	Set/Get	TC/B, TC/E, TC/J, TC/K, TC/N, TC/R, TC/S, TC/T
	CJC/format	Get	CJC/DEGC, CJC/DEGF, CJC/KELVIN
	STATUS	Get	BUSY, ERROR, READY
	OFFSET	Get	Floating point numeric
	SLOPE	Get	Floating point numeric
	VALUE	Get	Unsigned integer numeric

Hardware features

- One analog input channel, numbered 0
- Supports thermocouple types B, E, J, K, N, R, S, and T
- Possible gain ranges:
  - ±146.25 mV
  - ±73.125 mV
- 512 bytes of nonvolatile FLASH program memory; used for storing configuration information, calibration data, and user data.

USB-7202

Use the components below to set or get device properties.

Component	Supported Property/Command	Set/Get	Supported Values
DEV	MFGSER	Get	Up to 8 numeric characters
	FWV	Get	MM.mm (M = major; m = minor)
	ID	Set/Get	Up to 56 characters
	MFGCAL	Get	yyyy-mm-dd HH:MM:SS
	MFGCAL{YEAR}	Get	Year as yyyy
	MFGCAL{MONTH}	Get	Month as mm
	MFGCAL{DAY}	Get	Day as dd
	MFGCAL{HOUR}	Get	0 - 59
	MFGCAL{MINUTE}	Get	0 - 59
	MFGCAL{SECOND}	Get	0 - 59
	FLASHLED		0 - 255
AI	RANGE{ch}	Set/Get	BIP10V, BIP5V, BIP2V, BIP1V
	OFFSET	Get	4-byte floating point numeric
	SLOPE	Get	4-byte floating point numeric
	VALUE	Get	A/D range
	VALUE/format	Get	RAW, VOLTS
	CAL	Set/Get	ENABLE, DISABLE
	SCALE	Set/Get	ENABLE, DISABLE
AISCAN	RANGE{ch}	Set/Get	BIP10V, BIP5V, BIP2V, BIP1V
	RANGE	Set	Sets all channels to specified range
	XFRMODE	Set/Get	BLOCKIO, SINGLEIO
	RATE	Set/Get	0.596 to 50,000 Hz (1 channel)
	SAMPLES	Set/Get	0 to N (0 = continuous scan; N = 32-bit)
	HIGHCHAN	Set/Get	0 - 7
	LOWCHAN	Set/Get	0 - 7 (must be ≤ HIGHCHAN)
	EXTPACER	Set/Get	ENABLE/MASTER, ENABLE/SLAVE, DISABLE
	DEBUG	Set/Get	ENABLE, DISABLE
	STATUS	Get	IDLE, RUNNING, OVERRUN
	START		
	STOP		
	CAL	Set/Get	ENABLE, DISABLE
	SCALE	Set/Get	ENABLE, DISABLE
AITRIG	TYPE	Set/Get	EDGE/RISING, EDGE/FALLING
DIO	DIR	Set	IN, OUT
		Get	0 – 255 (bit field: 0 = all output, 255 = all input)
	VALUE	Get	0 – 255 (port) 0 – 1 (bit)
CTR	VALUE	Set/Get	0 (Set) 0 – 4,294,967,295 (Get)
	START		

	STOP		

Hardware features

- One digital port. All bits are individually configurable as input or output.
- Eight analog input channels, numbered 0 - 7.
- Possible gain ranges:
  - ±10V
  - ±5V
  - ±2V
  - ±1V
- External trigger input
- External pacer input / output. This feature allows multiple devices on a single USB to acquire synchronized samples. One master device is used to drive the signal. Additional devices must be configured as slave devices using the "AISCAN:EXTPACER=*value*" message. Value may be "ENABLE[/MASTER]", "ENABLE[/SLAVE]" or "DISABLE".
- 1024 bytes of nonvolatile EEPROM memory; used for storing configuration information, calibration data, and user data.
- *RATE* takes a float value. If the input scan rate requested is less than the slowest rate supported by the device, the device is set to the slowest rate supported by the device. If the input scan rate requested is greater than the fastest rate supported by the device, the device is set to the maximum rate supported by the device.

USB-7204

Use the components below to set or get device properties.

Component	Supported Property/Command	Set/Get	Supported Values
DEV	MFGSER	Get	Up to 8 numeric characters
	FWV	Get	MM.mm (M = major; m = minor)
	ID	Set/Get	Up to 56 characters
	MFGCAL	Get	yyyy-mm-dd HH:MM:SS
	MFGCAL{YEAR}	Get	Year as yyyy
	MFGCAL{MONTH}	Get	Month as mm
	MFGCAL{DAY}	Get	Day as dd
	MFGCAL{HOUR}	Get	0 - 59
	MFGCAL{MINUTE}	Get	0 - 59
	MFGCAL{SECOND}	Get	0 - 59
	FLASHLED		0 - 255
AI	RANGE{ch}	Set/Get	BIP20V, BIP10V, BIP5V, BIP4V, BIP2PT5V, BIP2V, BIP1PT25V, BIP1V
	CHMODE	Set	SE, DIFF
	OFFSET	Get	4-byte floating point numeric
	SLOPE	Get	4-byte floating point numeric
	VALUE	Get	A/D range
	VALUE/format	Get	RAW, VOLTS
	CAL	Set/Get	ENABLE, DISABLE
	SCALE	Set/Get	ENABLE, DISABLE
AISCAN	RANGE{element/ch}	Set	Element: 0 - 15 Channel: 0-7 single-ended, 0-3 differential Range: BIP20V, BIP10V, BIP5V, BIP4V, BIP2PT5V, BIP2V, BIP1PT25V, BIP1V
	RANGE{ch}	Set/Get	See the range values above.
	XFRMODE	Set/Get	BLOCKIO, SINGLEIO
	RATE	Set/Get	0.596 to 50,000 Hz (1 channel)
	SAMPLES	Set/Get	0 to N (0 = continuous scan; N = 32-bit)
	HIGHCHAN	Set/Get	0 - 7
	LOWCHAN	Set/Get	0 - 7 (must be ≤ HIGHCHAN)
	EXTPACER	Set/Get	ENABLE/MASTER, ENABLE/SLAVE, ENABLE/GSLAVE
	QUEUE	Set/Get	ENABLE, DISABLE, RESET
	DEBUG	Set/Get	ENABLE, DISABLE
	STATUS	Get	IDLE, RUNNING, OVERRUN
	QUEUE	Set/Get	ENABLE, DISABLE, RESET
	START		
	STOP		
	CAL[2]	Set/Get	ENABLE, DISABLE
	SCALE[2]	Set/Get	ENABLE, DISABLE
AITRIG	Type	Set/Get	EDGE/RISING, EDGE/FALLING
	REARM	Set/Get	ENABLE, DISABLE

AO	VALUE	Set	0 to 4095
	SCALE[2]	Set/Get	ENABLE, DISABLE
DIO	DIR	Set/Get	IN, OUT (port-configurable)
	VALUE	Get	0 – 255 (port) 0 – 1 (bit)
CTR	VALUE	Set/Get	0 (Set) 0 – 4,294,967,295 (Get)
	START		
	STOP		

Hardware features

- Two digital ports. Each port is individually configurable as input or output.
- Eight analog input channels, numbered 0 - 7.
- Analog input mode is configurable for single-ended (eight channels) or differential (four channels).
- Possible gain ranges:
  - ±20V (differential mode)
  - ±10V (differential or single–ended mode)
  - ±5V (differential mode)
  - ±4V (differential mode)
  - ±2.5V (differential mode)
  - ±2V (differential mode)
  - ±1.25V (differential mode)
  - ±1V (differential mode)
- External trigger input
- External pacer input / output. This feature allows multiple devices to acquire synchronized samples. One master device is used to drive the signal. Additional devices must be configured as slave devices using the "AISCAN:EXTPACER=*value*" message. Value may be "ENABLE/MASTER," "ENABLE/SLAVE," or "ENABLE/GSLAVE."
  - When set to *ENABLE/SLAVE*, the first clock pulse after setting up the scan is ignored to ensure adequate setup time for the first conversion. Use this mode when the device is paced from a continuous clock source.
  - When set to *ENABLE/GSLAVE*, the first clock pulse after setting up the scan is held off to ensure adequate setup time for the first conversion. No pulses are ignored. Use this mode when the device is paced from another USB-7204.
- 1024 bytes of nonvolatile EEPROM memory; used for storing configuration information, calibration data, and user data.
- *RATE* takes a float value. If the input scan rate requested is less than the slowest rate supported by the device, the device is set to the slowest rate supported by the device. If the input scan rate requested is greater than the fastest rate supported by the device, the device is set to the maximum rate supported by the device.