DATA TRANSLATION

UM-22024-N

Data Translation
DAQ Adaptor for MATLAB®

Thirteenth Edition
December, 2016

Data Translation, Inc.
100 Locke Drive
Marlboro, MA 01752-1192
(508) 481-3700
www.datatranslation.com
Fax: (508) 481-8620
E-mail: info@datx.com

Table of Contents

About this Manual

This manual describes how to use the Data Translation DAQ Adaptor for MATLAB to program Data Translation DT-Open Layers devices using MATLAB® from the Mathworks.

Intended Audience

This document is intended for engineers, scientists, technicians, or others responsible for using and/or programming DT-Open Layers-compliant devices using MATLAB. It is assumed that you have some familiarity with data acquisition principles and that you understand your application.

How this Manual is Organized

The manual is organized as follows:

- Chapter 1, "Overview," describes the Data Translation DAQ Adaptor for MATLAB, describes the session-based and legacy interfaces of the DAQ Adaptor for MATLAB, and provides references to related information.
- Chapter 2, "Session-Based DAQ Adaptor for MATLAB," describes the session-based Data Translation DAQ Adaptor for MATLAB.
- Chapter 3, "Legacy DAQ Adaptor for MATLAB," describes the legacy Data Translation DAQ Adaptor for MATLAB.

Conventions Used in this Manual

The following conventions are used in this manual:

- Notes provide useful information or information that requires special emphasis, cautions provide information to help you avoid losing data or damaging your equipment, and warnings provide information to help you avoid catastrophic damage to yourself or your equipment.
- Code snippets are shown in `courier`.

Where To Get Help

Should you run into problems installing or using the Data Translation DAQ Adaptor for MATLAB, the Data Translation Technical Support Department is available to provide technical assistance. Refer to the Data Translation web site (www.datatranslation.com) or call 508-946-5100. If you are outside the United States or Canada, call your local distributor, whose number is listed on our web site (www.datatranslation.com).

Overview

Overview

The Data Translation DAQ Adaptor for MATLAB provides an interface between MATLAB and the Data Acquisition Toolbox and Data Translation's DT-Open Layers architecture. Figure 1 shows the data flow model using the Data Translation DAQ Adaptor for MATLAB.

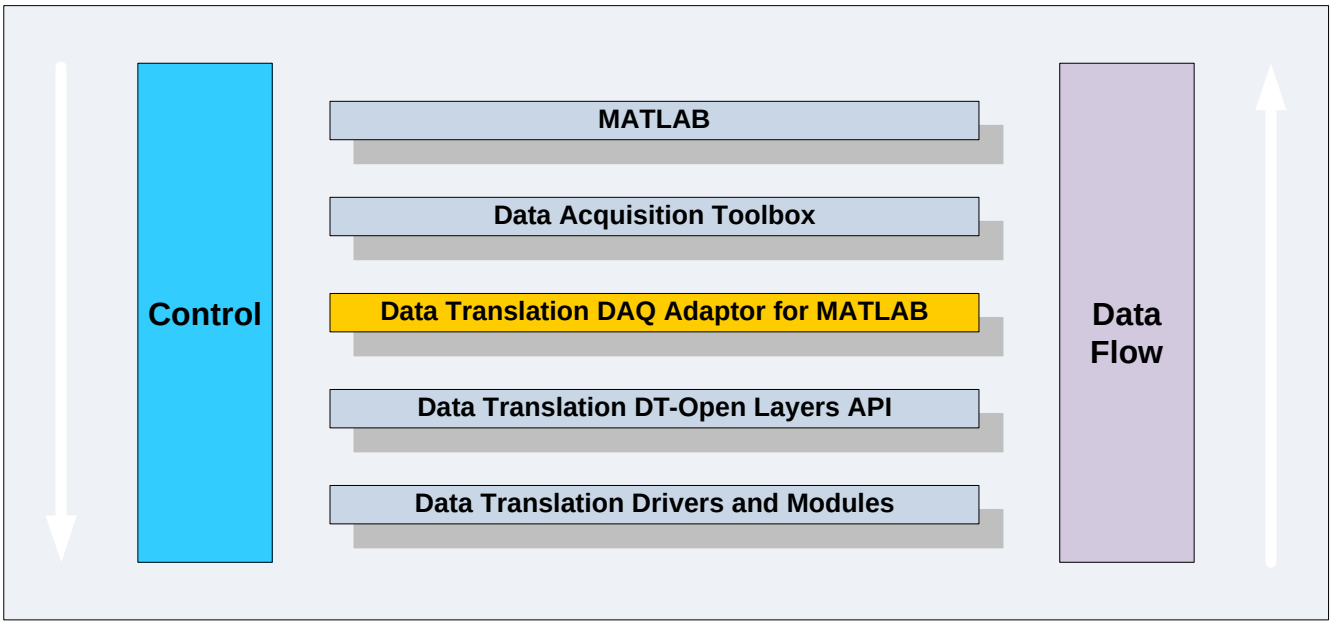

Figure 1: Data Flow Model Using the Data Translation DAQ Adaptor for MATLAB

DT-Open Layers provides an interface to all of Data Translation's analog and digital I/O data acquisition hardware modules. By using the Data Translation DAQ Adaptor for MATLAB with MATLAB and the Data Acquisition Toolbox, you can acquire data from Data Translation modules directly within MATLAB and then use all of MATLAB's features to analyze and display the data.

Session-Based or Legacy Interface

Data Translation provides both a 64-bit session-based interface and a 32-bit legacy interface of the DAQ Adaptor for MATLAB.

If want to use a 64-bit version of MATLAB (release R2016a or greater of MATLAB), you need the session-based DAQ Adaptor for MATLAB.

For all previous versions of MATLAB (32-bit), you need the legacy DAQ Adaptor for MATLAB.

Both interfaces are described in this manual.

Related Information

Refer to your MATLAB documentation, including documentation for the Data Acquisition toolbox, and the user's manuals for your Data Translation modules. For more information on DT-Open Layers, refer to following documents from Data Translation:

- Online help for the Data Acquisition Toolbox for MATLAB: https://www.mathworks.com/help/daq
- *DataAcq SDK User's Manual* (UM-18326). For programmers who are developing their own application programs using the Microsoft C compiler, this manual describes how to use the DT-Open Layers™ DataAcq SDK™ to access the capabilities of Data Translation data acquisition devices.
- *DT-Open Layers for .NET User's Manual* (UM-22161). For programmers who are developing their own application programs using Visual C# or Visual Basic .NET, this manual describes how to use the DT-Open Layers for .NET Class Library to access the capabilities of Data Translation data acquisition devices.

Session-Based DAQ Adaptor for MATLAB

Overview

Data Translation's session-based DAQ Adaptor for MATLAB supports analog and digital I/O operations on Data Translation DT-Open Layers-compliant modules.

Note: For Data Translation modules only, if your device supports digital input, tachometer, or counter operations in the analog input stream, you can read the value of the digital input port, tachometer, or counter channel through the specified analog input subsystem using the session-based DAQ Adaptor for MATLAB.

Similarly, if your Data Translation device supports digital output operations in the analog output stream, you can update the value of the digital output port through the specified analog output subsystem using the session-based DAQ Adaptor for MATLAB.

MEASURpoint instruments are not supported using the session-based DAQ Adaptor for MATLAB.

Requirements

You must install the following before installing the session-based DAQ Adaptor for MATLAB:

- Your Data Translation hardware module(s) and their drivers (from the Data Acquisition OMNI CD that ships with your hardware)
- A 64-bit version of MATLAB - Version R2016a or higher
- Data Acquisition Toolbox Version 3.9 or higher

Install Data Translation components from a recent OMNI CD, or download the software from our web site (www.datatranslation.com). Contact The MathWorks for updates to your MATLAB components.

Installation

You can install the session-based DAQ Adaptor for MATLAB one of the following ways:

From the Data Translation web site:

1. Navigate to the following web page for the Data Translation DAQ Adaptor for MATLAB: http://www.datatranslation.com/Products/Data-Acquisition-Software/MATLAB-Data-Acquisition
2. Click the **Download** tab.
3. Click **Free Download**.
4. Click **Session-Based DAQ Adaptor for MATLAB**, and then click **Download**.
   *The file **Data Translation Data Acquisition Toolbox.mltbx** is downloaded to your computer.*
5. Once it is downloaded, right-click on the file **Data Translation Data Acquisition Toolbox.mltbx**, and click **Install**.
   The Install Data Translation Data Acquisition Toolbox dialog appears.
6. Click **Install**.
   The Data Translation DAQ Adaptor for MATLAB is installed.

From within MATLAB:

1. Click **Add-Ons.**
   The Add-Ons Explorer appears.
2. Search for **Data Translation**.
3. Click **Session-Based DAQ Adaptor for MATLAB**.
4. Click **Add**, and select **Add to MATLAB**.
   The Data Translation DAQ Adaptor for MATLAB is installed.

From the MATLAB Central web site:

1. Navigate to the MATLAB Central main page:
   http://www.mathworks.com/matlabcentral/?s_tid=srchtitle
2. Search for **Data Translation**.
3. Click **Session-Based DAQ Adaptor for MATLAB**.
4. Click **Download Zip**.
   *The file **Data Translation Data Acquisition Toolbox.mltbx** and the PDF file that contains the documentation are downloaded to your computer.*
5. Once downloaded and unzipped, right-click on the file **Data Translation Data Acquisition Toolbox.mltbx**, and click **Install**.
   The Install Data Translation Data Acquisition Toolbox dialog appears.
6. Click **Install**.
   The Data Translation DAQ Adaptor for MATLAB is installed.

Hardware Discovery and Session Setup

Refer to the following web page for information on the functions and properties that are available for hardware discovery and session setup:
http://www.mathworks.com/help/daq/hardware-discovery-and-session-setup.html

Note that to create a data acquisition session for Data Translation devices, use the **daq.createSession** function and specify `'dt'` as the vendor name.

Analog Input Operations

This section describes information about performing analog input operations using the MATLAB session-based interface.

Analog Input Channel Functions, Events, and Properties

Refer to the following web page for information on the functions, events, and properties for setting up analog input channels:
http://www.mathworks.com/help/daq/analog-data-acquisition.html

For Data Translation devices, note the following differences:

- **TerminalConfig** – For Data Translation devices, the terminal configuration (SingleEnded or Differential) applies to all the channels of the analog input subsystem. Therefore, you cannot specify one channel as SingleEnded and another channel as Differential for the same subsystem.
- **Range** – Gains are not specified in MATLAB. Therefore, when specifying the input range for an analog input channel on a Data Translation device, you must specify the effective input range that you desire. For example, assume that your device supports an input range of ±10 V and a gain of 1, which yields an effective input range of ±10 V, and a gain of 2 , which provides an effective input range of ±5 V. For the **Range** property, specify either –10 V to +10 V or –5 V to +5 V for the channel.

  Some devices also allow you to read the value of other input channels, such as the digital input port, tachometer, or counters, through the analog input stream. For these devices, a range appropriate to that channel type, such as 0 to 65535 for the digital input port, is provided. In these cases, the input range applies only to the specified channel type.
- **MeasurementType** – Currently, `'Voltage'` (input) and `'IEPE'` (input) measurement types are available for Data Translation devices that support these capabilities.

Setting up an Analog Input Operation

This section shows an example of setting up an analog input operation for a Data Translation device using the session-based DAQ Adaptor for MATLAB:

1. Discover connected devices using the **daq.getDevices** function. In the following example, the system has several devices installed, including the Data Translation DT9837B device. The DT9837B device has the ID `'DT9837-B(01)'`, which is set in the Open Layers Control Panel (see the documentation for your hardware for more information on the Open Layers Control Panel):

```
devs = daq.getDevices
ans =
Data acquisition devices:
index Vendor Device ID             Description
----- ------ ---------  -------------------------------
1     ni       Dev1    National Instruments USB-9201
2     ni       Dev2    National Instruments USB-9234
3     ni       Dev4    National Instruments USB-4431
4     ni       Dev5    National Instruments USB-9265
5 directsound  Audio0  DirectSound Primary Sound Capture Driver
6 directsound  Audio1  DirectSound Microphone (Realtek High Definition Audio)
7 directsound  Audio2  DirectSound Primary Sound Driver
8 directsound  Audio3  DirectSound Speaker/HP Realtek High Definition Audio)
9    dt      DT9837-B(01) Data Translation DT9837-B(01)
```

2. To get information about the device, click the device ID or specify the index of the device to query. For example, to get information about the Data Translation device with index 9, use the following command:

```
devs(9)
```

   The following information is returned:

```
dt.Data Translation DT9837-B(01) (Device ID:'DT9837-B(01)')
Analog input subsystem supports:
-10 to +10 Volts, -1.0 to +1.0 Volts ranges
Rates from 195.3 to 105469.0 scans/sec
7 channels ('0'-'6')
'Voltage' measurement type
```

3. Create a session for Data Translation devices using the **daq.createSession** function, specifying `'dt'` as the vendor name. An example follows; the default number of scans and the acquisition duration are shown:

```
s = daq.createSession('dt')
s
s =
Data acquisition session using Data Translation hardware:
Will run for 1 second (1000 scans) at 1000.116 scans/second.
No channels have been added.
```

   By default, the acquisition is configured to run for a duration of 1 second to acquire 1000 scans; for this device, the default rate is 1000.116 scans per second.

4. Configure the session, as needed, using the following properties: **NumberofScans**, **DurationinSeconds**, **Rate**, or **IsContinuous**. The following example sets the duration of the acquisition to 10 seconds:

```
s.DurationInSeconds = 10;
s
s =
Data acquisition session using Data Translation hardware:
Will run for 10 seconds (10000 scans) at 1000.116 scans/second.
No channels have been added.
```

5. Add analog input channels to the session using the **addAnalogInputChannel** function. The following example adds analog input channels 0 and 1 of the DT9837-B(01) device to the session; note that these channels are configured for the IEPE measurement type:

```
ch0 = s.addAnalogInputChannel('DT9837-B(01)','0', 'IEPE');
ch1 = s.addAnalogInputChannel('DT9837-B(01)','1', 'IEPE');
s
s=
Data acquisition session using Data Translation hardware: Will run
for 10 seconds (10000 scans) at 1000.116 scans/second.
Number of channels: 2
index Type Device        Channel MeasurementType Range      Name
_____ ____ ____________ _______ _______________ __________ _____
1     ai   DT9837-B(01)  0       IEPE (Diff)  -10 to +10 Volts
2     ai   DT9837-B(01)  1       IEPE (Diff)  -10 to +10 Volts
```

Note: If desired, you can add analog input and analog output channels from one or multiple devices in the same session.

6. Configure the analog input channels, as needed, using the following properties: **Range**, **TerminalConfig**, **ID**, **Coupling**, **ExcitationCurrent,** and/or **ExcitationSource**.

   The following example configures IEPE inputs for analog input channel 0 (*ch* index 1) and analog input channel 1 (*ch* index 2) of the 'DT9837-B(01)' device. Channel 0 is configured for AC coupling and an internal excitation current value of 4 mA. Channel 1 is configured for DC coupling and no excitation source is used:

```
ch0.Range = [-10 10];
ch0.TerminalConfig='SingleEnded';
ch0.Coupling='AC';
ch0.ExcitationCurrentSource='Internal';
ch0.ExcitationCurrent = 0.004;
ch1.Range = [-10 10];
ch1.TerminalConfig='SingleEnded';
ch1.Coupling='DC';
ch1.ExcitationCurrentSource='None';
```

Adding Other Channels in the Analog Input Stream

If your Data Translation device supports reading the tachometer, digital input port, or counter through the analog input stream, you can add these as channels using the **addAnalogInputChannel** function. Note that you must specify `'Voltage'` as the measurement type using the **addAnalogInputChannel** function; however, in these cases, the value is treated as a raw number rather than a voltage.

Refer to the user's manual for your device to determine whether other channels can be added to the analog input stream as well as their corresponding channel numbers.

The following example adds the tachometer (channel 4) of the DT9837B device with device ID `'DT9837-B(01)'` to the analog input stream:

```
ch = s.addAnalogInputChannel('DT9837-B(01)','4', 'Voltage');
```

Get information about the channel:

```
ch
ans =

Data acquisition analog input voltage '4' on device 'DT9837-B(01)'
Terminal Config: Differential
Range: 0 to 4294967295
Name:"
Device:[1x1 daq.dt.DeviceInfo]
MeasurementType:'Tachometer'
```

Note that the range for the channel depends on the channel type. For example, the range for a 16-bit digital input port is 0 to 65535. The value that is returned represents the value of the channel, so in this example, if the first eight bits are set to 1 and the other bits are set to 0, then a value of 255 would be returned for this channel.

Note that if you add the analog input channels as well as other channels to the input stream, the specified voltage range is used for the analog input channels only.

Performing a Single-Value Input Operation

You can acquire a single analog input value using the **inputSingleScan** function.

The following example shows how to acquire a single analog input value for a Data Translation device using the session-based DAQ Adaptor for MATLAB. A Data Translation DT9837B device with device ID `'DT9837-B(01)'` is used in this example:

1. Create a session:

   ```
   s = daq.createSession('dt');
   ```

2. Add analog input channels 0 and 1 to the session:

   ```
   s.addAnalogInputChannel('DT9837-B(01)','0', 'Voltage');
   s.addAnalogInputChannel('DT9837-B(01)','1', 'Voltage');
   ```

3. Acquire an analog input value for each input channel that was added:

   ```
   data = inputSingleScan(s);
   ```

   One value is acquired for each analog input channel and the operation stops.

4. Read the acquired data. In this example, the value –0.1495 V is acquired from analog input channel 0 and 0.8643 V is acquired from analog input channel 1.

   ```
   data =
       -0.1495    0.8643
   ```

Performing a Foreground Acquisition

When you perform an analog input operation in the foreground, the operation blocks MATLAB until the operation is complete. Therefore, no other MATLAB commands can be used until the operation completes.

The following example shows how to start an analog input operation in the foreground. A Data Translation DT9837B device with device ID `'DT9837-B(01)'` is used in this example:

1. Create a session and add analog input channel 0 to the session:

   ```
   s = daq.createSession('dt');
   ```

2. Add analog input channel 0 to the session:

   ```
   s.addAnalogInputChannel('DT9837-B(01)', 0, 'IEPE');
   ```

3. Set the sample rate to 10 kHz:

   ```
   s.Rate = 10000;
   ```

4. Set the channel's ExcitationCurrent to 4 mA:

   ```
   s.Channels(1).ExcitationCurrent = .004;
   ```

5. Start the analog input session in the foreground and save the data and the acquisition time in the variables *[data, time]*:

   ```
   [data,time] = startForeground(s);
   ```

6. Plot the data:

   ```
   plot(time, data);
   ```

7. When you are finished with the operation, delete the session:

```
delete (s);
```

Performing a Background Acquisition

When you perform an analog input operation in the background, other MATLAB commands can be executed while the analog input operation is in progress. MATLAB is not blocked. A background acquisition depends on events and listeners to allow your code to access data as the hardware acquires it and to react to any errors as they occur.

Refer to the following web page for more information on events and listeners in background operations: http://www.mathworks.com/help/matlab/matlab_oop/events-and-listeners--concepts.html

To start an analog input operation in the background, use the **startBackground** command. Use the **DataAvailable** event to determine when the acquired data is available.

Note: By default the **DataAvailable** event fires when 1/10 second worth of data is available for analysis. If you want to change the frequency of when the **DataAvailable** event is fired, set **IsNotifyWhenDataAvailableExceedsAuto** to false, and modify the value of **NotifyWhenDataAvailableExceeds** as desired. Refer to your MATLAB documentation for more information.

Listeners execute a callback function when notified that the event has occurred. Use **Session.addlistener** to create a listener object that executes your callback function.

By default, the **IsContinuous** property is set to False and the operation stops automatically. If you have a continuous operation, set **IsContinuous** to True and use **stop** when you want to stop the background operation.

If desired, you can use **wait** to block MATLAB execution until the background operation is complete.

This example shows how to acquire data from a Data Translation DT9837B device with device ID `'DT9837-B(01)'` in the background using events and listeners:

1. Create an Data Translation session object:

```
s = daq.createSession('dt');
```

2. Add analog input 0 to the session:

```
addAnalogInputChannel(s,'DT9837-B(01)', '0', 'Voltage');
```

3. Add the listener for the DataAvailable event and assign it to the variable *lh*:

```
lh = addlistener(s,'DataAvailable', @plotData);
```

4. Create a simple callback function to plot the acquired data and save it as *plotData.m* in your working directory:

```
function plotData(src,event)
  plot(event.TimeStamps, event.Data)
end
```

   Here, *src* is the session object for the listener and event is a *daq.DataAvailableInfo* object that contains the data and associated timing information.

5. Start a continuous background operation and see the plot update while MATLAB is running:

```
s.IsContinuous = true;
startBackground(s);
```

6. When you are finished acquiring data, stop the operation:

```
s.stop();
```

7. Delete the listener:

```
delete (lh);
```

8. When you are finished with the operation, delete the session:

```
delete (s);
```

Triggering Analog Input Operations

Note: Refer to the following web page for information on the functions and properties for setting up triggers using MATLAB:
http://www.mathworks.com/help/daq/simultaneous-and-synchronized-operations.html

You can trigger analog input operations based on an external event if your Data Translation device supports an external TTL trigger for the analog input subsystem. Refer to the user's manual for your device to determine its supported capabilities.

To specify an external TTL trigger on a Data Translation device, use the MATLAB **addTriggerConnection** function, and specify the following parameters:

- For the trigger source, specify `'external'`.
- For the trigger destination, specify 'deviceID/EXT_TTL_AI_TRIG', where *deviceID* is the ID of your device as defined in the Open Layers Control Panel. See the documentation for your hardware for more information on pin assignments and the Open Layers Control Panel.
- For the trigger type, specify 'StartTrigger'.

Once you have set up the trigger connection, use the **TriggerCondition** property to set the trigger condition to `'RisingEdge'` or `'FallingEdge'`.

The following example shows how to trigger an analog input operation on a DT9837B device with device ID `'DT9837-B(01)'` using a signal connected to the external TTL trigger input on the device:

1. Create a session and add two analog input channels:

   ```
   s = daq.createSession('dt');
   ch = addAnalogInputChannel(s,'DT9837-B(01)',0:1,'Voltage');
   ```

2. Configure the terminal configuration and range of the channels in the session:

   ```
   ch(1).TerminalConfig = 'SingleEnded';
   ch(1).Range = [-10.0 10.0];
   ch(2).TerminalConfig = 'SingleEnded';
   ch(2).Range = [-10.0 10.0];
   ```

3. Create an external trigger connection and specify that a rising-edge trigger will trigger acquisition:

   ```
   addTriggerConnection(s,'external','DT9837-B(01)/EXT_TTL_AI_TRIG',
     'StartTrigger');
   s.Connections(1).TriggerCondition = 'RisingEdge';
   ```

4. Set the rate and the duration of the acquisition:

   ```
   s.Rate = 50000;
   s.DurationInSeconds = 0.01;
   ```

5. Set up the foreground operation:

   ```
   [data,timestamps] = startForeground(s);
   plot(timestamps,data);
   ```

   When the rising-edge trigger is detected on the external TTL trigger input pin on the DT9837-B(01) device, analog input data is acquired and plotted in the foreground.

Analog Output Operations

This section describes information about performing analog output operations using the MATLAB session-based interface.

Analog Output Channel Functions, Events, and Properties

Refer to the following web page for information on the functions, events, and properties for setting up analog input channels:
http://www.mathworks.com/help/daq/analog-signal-generation.html

For Data Translation devices, note the following differences:

- **TerminalConfig** – For Data Translation devices, the terminal configuration (SingleEnded or Differential) applies to all the channels of the analog output subsystem. This setting is typically not configurable for analog output subsystems but is returned.
- **Range** – One or more analog output voltage ranges may be supported by your Data Translation device. The voltage range applies to all analog output channels of the analog output subsystem.

  Some devices also allow you to update the digital output port through the analog output stream. For these devices, a digital output range, such as 0 to 65535, is provided. The digital output range applies only to the digital output channel in the analog output stream.
- **MeasurementType** – Currently, `'Voltage'` (output) is the only measurement type available for Data Translation devices. You must specify this measurement type even if you are setting up a digital output channel in the analog output stream.

Setting up an Analog Output Operation

This section shows an example of setting up an analog output operation for a Data Translation device using the session-based DAQ Adaptor for MATLAB:

1. Discover connected devices using the **daq.getDevices** function. In the following example, the system has several devices installed, including the Data Translation DT9836-12-2 device. The DT9836-12-2 device has the ID `'dt9836'`, which is set in the Open Layers Control Panel (see the documentation for your hardware for more information on the Open Layers Control Panel):

```
devs = daq.getDevices
ans =
Data acquisition devices:
index Vendor Device ID             Description
----- ------ ---------  -------------------------------
1 dt           dt9836  Data Translation DT9836-12-2
2 directsound  Audio0  DirectSound Primary Sound Capture Driver
3 directsound  Audio1  DirectSound Microphone (Realtek High Definition Audio)
4 directsound  Audio2  DirectSound Primary Sound Driver
5 directsound  Audio3  DirectSound DELL U2414H (Intel(R) Display Audio)
6 directsound  Audio4  DirectSound Speakers/Headphones (Realtek High Def Audio)
7 ni           Dev1    National Instruments USB-9234
8 ni           Dev2    National Instruments PCI-MIO-16XE-50
```

2. To get information about the device, click the device ID or specify the index of the device to query. For example, to get information about the Data Translation device with index 1, use the following command:

   ```
   devs(1)
   ```

   The following information is returned:

   ```
   dt.Data Translation DT9836-12-2 (Device ID:'dt9836')
    Analog input subsystem supports:
     -10 to +10 Volts, -5.0 to +5.0 Volts ranges
     Rates from 0.0 to 225000.0 scans/sec
     23 channels ('0' - '22')
     'Voltage' measurement type

    Analog output subsystem supports:
     -10 to +10 Volts ranges
     Rates from 0.0 to 500000.0 scans/sec
     3 channels ('0','1','2')
     'Voltage' measurement type
   ```

3. Create a session for Data Translation devices using the **daq.createSession** function, specifying '**dt**' as the vendor name. An example follows; the default number of scans and the acquisition duration are shown:

   ```
   s = daq.createSession('dt')
   s
   s =
   Data acquisition session using Data Translation hardware:
   Will run for 1 second (1000 scans) at 1000 scans/second.
   No channels have been added.
   ```

   By default, the operation is configured to run for a duration of 1 second to output If1000 scans; for this device, the default rate is 1000 scans per second.

4. Configure the session, as needed. For analog output operations, use the following properties: **DurationinSeconds**, **Rate**, and/or **IsContinuous**. The following example sets the output rate to 8 kHz:

   ```
   s.Rate = 8000;
   ```

5. Add analog output channels to the session using the **addAnalogOutputChannel** function. The following example adds analog output channels 0 and 1 of the dt9836 device to the session:

   ```
   ch0 = s.addAnalogOutputChannel('dt9836','0', 'Voltage');
   ch1 = s.addAnalogOutputChannel('dt9836','1', 'Voltage');
   ```

Note: If desired, you can add analog input and analog output channels from one or multiple devices in the same session.

Adding a Digital Output Port in the Analog Output Stream

If your Data Translation device supports updating the digital output port through the analog output stream, you can add the digital output port as a channel using the **addAnalogOutputChannel** function. Note that you must specify `'Voltage'` as the measurement type using the **addAnalogOutputChannel** function even though it is a digital output channel. The DAQ Adaptor for MATLAB ignores voltage for this channel, treating the data as a raw value.

The following example adds the digital output port (channel 2) of the DT9836-12-2 device (device ID `'dt9836'`) to the analog output stream:

```
aoutch = s.addAnalogOutputChannel('dt9836','2', 'Voltage');
```

Get information about the channel:

```
aoutch
ans =

Data acquisition analog output voltage '2' on device 'dt9836'
Terminal Config: Differential
Range: 0 to +65535 DigitalOutput
Name:
Device:[1x1 daq.dt.DeviceInfo]
MeasurementType:'DigitalOutput'
```

The range for a digital output channel is 0 to 65535, which represents the 16-bit digital output port. Note that if you add the analog output channels as well as the digital output channel to the output stream, the specified voltage range is used for the analog output channels and the 0 to 65535 range is used for the digital output channel.

Note: The MeasurementType that is returned for the digital output channel in the analog output stream is `'DigitalOutput'`, which indicates the channel type. However, when using the **addAnalogOutputChannel** function, you must specify `'Voltage'` for this channel type.

When you want to update the digital output channel, specify a value that corresponds to the bits of the 16-bit digital port. For example, to set the lower eight bits to 1 and the higher eight bits to 0, specify a value of 255.

Performing a Single-Value Output Operation

You can output a single analog output value using the **outputSingleScan f**unction.

The following example shows how to output a single analog output value to a Data Translation device using the session-based DAQ Adaptor for MATLAB. A Data Translation DT9836-12-2 device (device ID `'dt9836'`) is used in this example:

1. Create a session:

   ```
   s = daq.createSession('dt');
   ```

2. Add analog output channels 0 and 1 to the session:

   ```
   s.addAnalogOutputChannel('dt9836', 0, 'Voltage');
   s.addAnalogOutputChannel('dt9836', 1, 'Voltage');
   ```

3. Create an output value and output a single scan for each output channel that was added.

   ```
   outputSingleScan(s,[1.5 4]);
   ```

   In this example, a value of 1.5 V is output to analog output channel 0 and a value of 4 V is output to analog output channel 1. One value is output for each channel and the operation stops.

Performing a Foreground Output Operation

When you perform an analog output operation in the foreground, the operation blocks MATLAB until the operation is complete. Therefore, no other MATLAB commands can be used until the operation completes.

The following example shows how to start an analog output operation in the foreground. A Data Translation DT9836-12-2 device with device ID `'dt9836'` is used in this example:

1. Create a session:

   ```
   s = daq.createSession('dt');
   ```

2. Add analog output channel 0 to the session:

   ```
   s.addAnalogOutputChannel('dt9836', 0, 'Voltage');
   ```

3. Create the data to output:

   ```
   outputData = linspace(-1, 1, 2200)';
   ```

4. Queue the data:

   ```
   queueOutputData(s,outputData);
   ```

   The duration changes based on the length of the queued data and the specified scan rate. When the session contains output channels, duration and number of scans become read-only properties of the session. The number of scans in a session is determined by the amount of data queued and the duration is determined by **s.ScansQueued** divided by **s.Rate**.

5. Generate the output signal in the foreground; MATLAB returns once the data has been output.

   ```
   startForeground(s);
   ```

6. When you are finished, delete the session:

```
delete (s);
```

Performing a Background Output Operation

When you perform an analog output operation in the background, other MATLAB commands can be executed while the analog output operation is in progress. MATLAB is not blocked. A background output operation depends on events and listeners to allow your code to continue to output data as the queue data is depleted and to react to any errors that may occur.

To start an analog output operation in the background, use the **startBackground** command. Use the **DataRequired** event to determine when more data needs to be queued to the device. This event is fired periodically when a continuous output operation is in progress.

Listeners execute a callback function when notified that the event has occurred. Use **addlistener** to create a listener object that executes your callback function.

Refer to the following web page for more information on events and listeners in background operations: http://www.mathworks.com/help/matlab/matlab_oop/events-and-listeners--concepts.html

By default, the **IsContinuous** property is set to False and the operation stops automatically. If you want to perform a continuous output operation, set **IsContinuous** to True and use **stop** when you want to stop the background operation.

To start an analog output operation in the background, use the **startBackground** command.

This example shows how to output analog data from a Data Translation DT9836-12-2 device (device ID `'dt9836'`) in the background:

1. Create an Data Translation session object:

```
s = daq.createSession('dt');
```

2. Add analog output 0 to the session:

```
addAnalogOutputChannel(s,'dt9836', '0', 'Voltage');
```

3. Set the output rate. In this example, the output rate is set to 8 kHz:

```
s.Rate = 8000;
```

4. Create the data to output:

```
data = linspace(0,pi,s.Rate)';
outputSignal = sin(data*2)*2;
```

5. Queue the output data:

```
queueOutputData(s,outputSignal);
```

6. Set up an output event handler that adds a listener for DataRequired events. In this example, the event handler requeues the output data when the DataRequired event occurs; the listener for the DataRequired event is assigned to the variable *lh*:

```
lh = addlistener(s,'DataRequired', ...
     @(src,event) src.queueOutputData(outputSignal));
```

7. Generate the output signal in the background:

```
s.IsContinuous = true;
startBackground(s);
```

   You can execute other MATLAB commands while the signal generation operation is in progress.

8. When you are finished generating the signal, stop the operation:

```
s.stop();
```

9. Delete the listener:

```
delete (lh);
```

10. When you are finished, delete the session:

```
delete (s);
```

Triggering Analog Output Operations

Note: Refer to the following web page for information on the functions and properties for setting up triggers using MATLAB:
http://www.mathworks.com/help/daq/simultaneous-and-synchronized-operations.html

You can trigger analog output operations based on an external event if the your Data Translation device supports an external TTL trigger for the analog output subsystem. Refer to the user's manual for your device to determine its supported capabilities.

To specify an external TTL trigger for an analog output operation on a Data Translation device, use the MATLAB **addTriggerConnection** function with the following parameters:

- For the trigger source, specify `'external'`.
- For the trigger destination, specify 'deviceID/EXT_TTL_AO_TRIG', where *deviceID* is the ID of your device as defined in the Open Layers Control Panel. See the documentation for your hardware for more information on pin assignments and the Open Layers Control Panel.
- For the trigger type, specify 'StartTrigger'.

Once you have set up the trigger connection, use the **TriggerCondition** property to set the trigger condition to `'RisingEdge'` or `'FallingEdge'`.

The following example shows how to trigger an analog output operation on a DT9836 device (device ID `'dt9836'`) using a signal connected to the external TTL trigger input on the device:

1. Create a session:

```
s = daq.createSession('dt');
```

2. Add analog output channel 0 to the session:

```
s.addAnalogOutputChannel('dt9836', 0, 'Voltage');
```

3. Create an external trigger connection and specify that a rising-edge trigger will trigger the analog output operation:

```
addTriggerConnection(s,'external','dt9836/EXT_TTL_AO_TRIG',
  'StartTrigger');
s.Connections(1).TriggerCondition = 'RisingEdge';
```

4. Create the data to output:

```
outputData = linspace(-1, 1, 2200)';
```

5. Queue the data:

```
queueOutputData(s,outputData);
```

6. Set up the foreground operation:

```
startForeground(s);
```

   When the rising edge is detected on the on external trigger pin of the DT9836 device, the analog output operation is started in the foreground. MATLAB returns once the data has been output.

Analog Input and Output Operations at the Same Time

The DAQ Adaptor for MATLAB supports performing analog input and analog output operations at the same time. For DT-Open Layers-compliant devices that support this feature, the DAQ Adaptor automatically starts both operations at the same time (unless different trigger sources are used for each subsystem).

Performing I/O Operations at the Same Time in the Foreground

The following example shows how to perform analog input and output operations at the same time in the foreground. This examples uses a Data Translation DT9836-12-2 device (device ID `'dt9836'`):

1. Create a Data Translation session object:

   ```
   s = daq.createSession('dt');
   ```

2. Add analog output 0 to the session:

   ```
   addAnalogOutputChannel(s,'dt9836', '0', 'Voltage');
   ```

3. Add analog input 0 to the session:

   ```
   addAnalogInputChannel(s,'dt9836', '0', 'Voltage');
   ```

4. Set the properties of the session. Note that for any given session, acquisition and signal generation run at the same rate. In this example, the acquisition and signal generation rate is set to 8 kHz:

   ```
   s.Rate = 8000;
   ```

5. Create the data to output:

   ```
   data = linspace(0,pi,s.Rate)';
   outputSignal1 = sin(data)*2;
   ```

6. Queue the output data:

   ```
   queueOutputData(s,outputSignal1);
   ```

7. Generate the output signal and acquire data in the foreground:

   ```
   [data,time] = startForeground(s);
   ```

   MATLAB returns once the I/O operations are complete.

8. Plot the data:

   ```
   plot(time, data);
   ```

9. When you are finished, delete the session:

   ```
   delete (s);
   ```

Performing I/O Operations at the Same Time in the Background

The following example shows how to perform analog input and output operations at the same time n the background. This examples uses a Data Translation DT9836-12-2 device (device ID `'dt9836'`):

1. Create a Data Translation session object:

   ```
   s = daq.createSession('dt');
   ```

2. Add analog output 0 to the session:

   ```
   addAnalogOutputChannel(s,'dt9836', '0', 'Voltage');
   ```

3. Add analog input 0 to the session:

   ```
   addAnalogInputChannel(s,'dt9836', '0', 'Voltage');
   ```

4. Set the properties of the session. Note that for any given session, acquisition and signal generation run at the same rate. In this example, the acquisition and signal generation rate is set to 8 kHz:

   ```
   s.Rate = 8000;
   ```

5. Create the data to output:

   ```
   data = linspace(0,pi,s.Rate)';
   outputSignal1 = sin(data)*2;
   ```

6. Queue the output data:

   ```
   queueOutputData(s,outputSignal1);
   ```

7. Set up an output event handler that adds a listener for DataRequired events. In this example, the event handler requeues the output data when the DataRequired event occurs; the listener for the DataRequired event is assigned to the variable *dataRequiredListener*:

   ```
   dataRequiredListener= addlistener(s,'DataRequired', ...
        @(src,event) src.queueOutputData(outputSignal1));
   ```

8. Set up an input event handler that adds a listener for DataAvailable events. In this example, the event handler plots the acquired data against time when the DataAvailable event occurs; the listener for the DataAvailable event is assigned to the variable *dataAvailableListener*:

   ```
   dataAvailableListener = addlistener(s,'DataAvailable',...
        @(src,event)plot(event.TimeStamps, event.Data));
   ```

9. Generate the output signal and acquire data in the background:

   ```
   s.IsContinuous = true;
   startBackground(s);
   ```

   You can execute other MATLAB commands while the signal generation operation is in progress.

10. When you are finished generating the output signal and acquiring input data, stop the operation:

    ```
    s.stop();
    ```

11. Delete the listeners:

```
delete (dataRequiredListener);
delete (dataAvailableListener);
```

12. When finished, delete the session:

```
delete (s);
```

Digital I/O Operations

This section describes information about performing digital I/O operations using the MATLAB session-based interface.

Refer to the following web page for information on the functions, events, and properties for performing digital I/O operations:
http://www.mathworks.com/help/daq/digital-input-and-output.html

Note: Data Translation devices do not support foreground or background digital I/O operations. In addition, interrupt-on-change operations are not supported by the session-based DAQ Adaptor for MATLAB.

Performing Digital Input Operations

If your Data Translation device supports reading the digital input port through the analog input stream, refer to page 19 for information about adding the digital input port as a channel of the analog input subsystem.

If your Data Translation device supports single-value digital input operations through the digital input subsystem, you can read a single value from one or more digital input lines using the **inputSingleScan** function of the session-based DAQ Adaptor for MATLAB.

The following example shows how to read a single value from lines 0 and 1 of digital input port 0 on a 'DT9857' device using the **inputSingleScan** function:

```
s = daq.createSession('dt');
addDigitalChannel(s,'DT9857', 'Port0/Line0:1', 'InputOnly');
data = inputSingleScan(s);
```

Performing a Digital Output Operation

If your Data Translation device supports updating the digital output port through the analog output stream, refer to page 26 for information about adding the digital output port as a channel to analog output subsystem.

If your Data Translation device supports a single-value digital output operation through the digital output subsystem, you can update one more digital output lines using the **outputSingleScan** function of the session-based DAQ Adaptor for MATLAB.

The following example shows how to output a value of 0 to line 0 and a value of 1 to line 1 of digital output port 0 on a 'DT9857' device using the **outputSingleScan** function:

```
s = daq.createSession('dt');
addDigitalChannel(s,'DT9857', 'Port0/Line0:1', 'OutputOnly');
outputSingleScan(s,[0 1])
```

Legacy DAQ Adaptor for MATLAB

Requirements

You must install the following before installing the legacy Data Translation DAQ Adaptor for MATLAB:

- Your Data Translation hardware module(s) and their drivers (from the Data Acquisition OMNI CD that ships with your hardware)
- A 32-bit version of MATLAB - Version 7 (R14) Service Pack 3 or higher
- Data Acquisition Toolbox Version 2.7 to 3.8

Install Data Translation components from a recent OMNI CD, or download from our web site (www.datatranslation.com). Contact The MathWorks for updates to your MATLAB components.

Installation

To install the legacy Data Translation DAQ Adaptor for MATLAB, do the following:

1. Navigate to the following web page for the Data Translation DAQ Adaptor for MATLAB: http://www.datatranslation.com/Products/Data-Acquisition-Software/MATLAB-Data-Acquisition
2. Click the **Download** tab.
3. Click **Free Download**.
4. Click **Legacy-Based DAQ Adaptor for MATLAB**, and then click **Download**.
   *The file **setup.exe** is downloaded to your computer.*
5. Once it is downloaded, double-click on the file **setup.exe**.
6. Close other programs if necessary, and click **Next** on the welcome screen.
7. Either change the directory path using **Browse** or accept the default directory, and then click **Next.**
   The installer asks you whether to back up replaced files.
8. Choose the **Yes** to back up replaced files or **No** if you do not wish to back up replaced files, and then click **Next.**
   The installer prompts you to begin file installation.
9. Click **Next.**
   The installer copies the files to the destination directory.

   The installer copies the files and then launches MATLAB. MATLAB registers the DAQ Adaptor .DLL using the path you provided:

   ```
   >>rehash toolboxcache;daqregister 'C:\PROGRA~1\DATATR~1\DAQADA~1\
       dtol.dll';quit
   >>
   ```

10. Click **Finish.**

Setting up an I/O Operation

The Data Translation legacy DAQ Adaptor for MATLAB is specified as **dtol** in MATLAB code. The following example shows how to create an analog input object (ai0) using the DTOL adaptor for a device at index 0:

```
ai0 = analoginput ('dtol', 0)
```

To add hardware channel 1 to the **ai0** object, use the MATLAB command **addchannel**, as follows:

```
addchannel(ai0, 1)
```

To get a single sample from that channel, use the MATLAB command **getsample,** as follows:

```
s = getsample(ai0)
```

You can display the properties of an adaptor (including installed hardware modules) with the **daqhwinfo** command. This example shows how to display the properties of the DTOL adaptor using the daqhwinfo command:

```
daqhwinfo('dtol')
```

If there are too many devices to display on a single line, you can assign **daqhwinfo** to an array and query the name of each device separately, as shown in this example:

```
x = daqhwinfo('dtol')
x.BoardNames
```

You can examine properties of a subsystem that you created with the adaptor by using the MATLAB **daqhwinfo** command (for built-in properties) and the `get` command (for user-settable properties). In this example, the first command shows all the built-in properties of the analog input subsystem (ai0) and the second command shows all the DTOL user-settable properties of the analog input subsystem (ai0):

```
daqhwinfo(ai0)
get(ai0)
```

You can find out more information about a property using the MATLAB **propinfo** command. For example, this command shows how to get information about the **GainPerChan** property that is associated with channel 1 of the analog input subsystem (ai0):

```
propinfo(ai0.channel(1),'GainPerChan')
```

To control a data acquisition session, issue MATLAB commands and set the appropriate DTOL properties, described in the following subsections. For more information about using MATLAB commands and the Data Acquisition Toolbox properties, refer to your MATLAB documentation.

Analog Input and Output Subsystem Properties

The Data Translation DAQ Adaptor for MATLAB provides the following adapter-specific properties for analog I/O operations:

Table 1: Adaptor-Specific Analog Input/Output Properties

Property	Description	A/D	D/A
FIFOAvailable	Determines if a FIFO is available on the subsystem.	No	Yes
FIFODepth	Determines the size of the FIFO.	No	Yes
OutOfDataMode	Specifies how to handle an out-of-data condition. Set *Hold* to use the last value in the output buffer. Set *Default* to use the channel-specific **Default Value** property.	No	Yes
SupportBinaryEncoding	Determines if the subsystem uses binary encoding.	Yes	Yes
SupportExternalTriggerFalling	Checks for external, falling-edge triggering support.	Yes	Yes
SupportExternalTriggerRising	Checks for external, rising-edge triggering support.	Yes	Yes
SupportGainPerChannel	Determines if the subsystem supports different gain values for each analog channel.	Yes	Yes
SupportSoftTrigger	Determines if the subsystem supports an internal software trigger.	Yes	Yes
SupportTwosComp	Determines if the subsystem uses twos complement encoding.	Yes	Yes
ReturnCjcTemperatureInStream[a]	Specifies whether to return CJC values in the analog input data stream.	Yes	No
SyncMode	Specifies the synchronization mode (None, Master, or Slave) for the analog input subsystem of the device.	Yes	No
DataFilterType[a]	Specifies the filter type (Raw or MovingAverage) for the analog input subsystem of the device.	Yes	No
ExcitationVoltageSource	Specifies the excitation voltage source (Internal or External) for the analog input subsystem of the device.	Yes	No
ExcitationVoltage	Specifies the excitation voltage for the excitation voltage source of the analog input subsystem.	Yes	No
WrapMode	Specifies how the analog output subsubsystem specifies buffers. Possible values are None (data is written from multiple output buffers continuously) or Single (data is written from a single buffer continuously). If the WrapMode is Single, the device operates in waveform mode and uses the first user buffer regardless of the number of MATLAB buffers that are set.	No	Yes

a. This property is visible only when a TEMPpoint or VOLTpoint instrument is specified.

Setting the Input Type

To specify whether a subsystem uses differential or single-ended input channels, use the MATLAB property **inputtype**. For example, this command configures the analog input subsystem (ai0) to use differential input channels:

```
ai0.inputtype='Differential'
```

To configure the analog input subsystem (ai0) to use single-ended channels, use this command:

```
ai0.inputtype='SingleEnded'
```

You can use the **get(ai0)** command to see the changes. When you change input types, the channel list also changes.

Setting the Synchronization Mode

Some devices provide a synchronization connector (such as a Sync Bus RJ45 connector) that allows you to synchronize operations on multiple devices. In this configuration, the subsystem on one device is configured as the master and the subsystem on the other device is configured as a slave. When the master module is triggered, both the master device and the slave device start operating at the same time.

If the device allows you to program the synchronization mode (SupportSynchronization is True), use the DTOL-specific property **SyncMode** to set the synchronization mode to one of the following values:

- None – The subsystem is configured to ignore the synchronization circuit on the device.
- Master – Sets the subsystem as a master; the synchronization connector on the device is configured to output a synchronization signal.
- Slave – Sets the subsystem as a slave; the synchronization connector on the device is configured to accept a synchronization signal as an input.

The following command sets the synchronization mode of the analog input subsystem to Master:

```
ai0.SyncMode='Master'
```

Setting the Filter Type

Some devices, such as TEMPpoint and VOLTpoint instruments, support a software-programmable filter type for the analog input subsystem. For these devices, the following filter types are available:

- Raw – No filter. Provides fast response times, but the data may be difficult to interpret. Use when you want to filter the data yourself.

  The Raw filter type returns the data exactly as it comes out of the Delta-Sigma A/D converters. Note that Delta-Sigma converters provide substantial digital filtering above the Nyquist frequency.

Generally, the only time it is desirable to turn off the software filter is if you are using fast responding inputs, sampling them at higher speeds (> 1 Hz), and need as much response speed as possible.

- MovingAverage – Provides a compromise of filter functionality and response time. This filter can be used in any application.

  This low-pass filter takes the previous 16 samples, adds them together, and divides by 16.

The following command sets the filter type of the analog input subsystem to MovingAverage:

```
ai0.DataFilterType='MovingAverage'
```

Channel Properties

Table 2 lists the properties that appear at the channel level after you add a channel. The following subsections describe each of these properties in more detail.

Table 2: Adaptor-Specific Channel Properties

Property	Description	A/D	D/A
MultiSensorType	For subsystems that support multiple sensor types for each channel, specify one of the following sensor types for each channel: VoltageIn, Current, Thermocouple, StrainGage, Accelerometer, Bridge, Thermistor, Resistance, RTD.	Yes	No
InputTerminationEnabled	For subsystems that support a bias return termination resistor, enables or disables the bias return termination resistor for a particular analog input channel.	Yes	No
Coupling	For subsystems that support IEPE (accelerometer) inputs, specify the coupling type (AC or DC) for a particular analog input channel.	Yes	No
ExcitationCurrentSource	For subsystems that support IEPE (accelerometer) and resistance inputs, specify the excitation current source for a particular analog input channel. Possible values are Internal (excitation current source is on the device), External (excitation current source is external to the device), or Disabled (no excitation is applied).	Yes	No
ExcitationCurrentValue	If you specify an internal excitation current source, specify the value of the current source.	Yes	No
GainPerChan	For subsystems that support multiple gain values, specify the gain value for a particular analog input channel.	Yes	No
ThermocoupleType[a]	For subsystems that support thermocouple measurements, specify the thermocouple type for a particular analog input channel. The following thermocouple types are supported: J,K,B,E,N,R,S,T, or None.	Yes	No

Table 2: Adaptor-Specific Channel Properties (cont.)

Property	Description	A/D	D/A
RtdType	For subsystems that support RTD measurements, specify the RTD type for a particular analog input channel. For MEASURpoint instruments, which use an IVI-COM device driver, the following RTD types are supported: Volts, Ohms, PT100_385, PT500_385, PT1000_385, PT100_392, PT500_392, or PT1000_392. For the DT9829 and other device that use a DT-Open Layers device driver, the following RTD types are supported: PT3750, PT3850, PT3911, PT3916, PT3920, PT3928, or Custom.	Yes	No
RtdR0	(Not supported by MEASURpoint instruments) For subsystems that support RTD measurements, specify the R0 coefficient for the RTD that is connected to a particular analog input channel. This coefficient is used by the Callendar-Van Dusen transfer function.	Yes	No
RtdA	(Not supported by MEASURpoint instruments) For subsystems that support RTD measurements, specify the A coefficient for the RTD that is connected to a particular analog input channel. This coefficient is used by the Callendar-Van Dusen transfer function.	Yes	No
RtdB	(Not supported by MEASURpoint instruments) For subsystems that support RTD measurements, specify the B coefficient for the RTD that is connected to a particular analog input channel. This coefficient is used by the Callendar-Van Dusen transfer function.	Yes	No
RtdC	(Not supported by MEASURpoint instruments) For subsystems that support RTD measurements, specify the C coefficient for the RTD that is connected to a particular analog input channel. This coefficient is used by the Callendar-Van Dusen transfer function.	Yes	No
SensorWiringConfiguration	(Not supported by MEASURpoint instruments) For subsystems that support RTD, thermistor, or resistance measurements, specify the wiring configuration (TwoWire, ThreeWire, or FourWire) used by a particular analog input channel.	Yes	No
ThermistorA	For subsystems that support thermistor measurements, specify the A coefficient for the thermistor that is connected to a particular analog input channel. This coefficient is used by the Steinhart-Hart transfer function.	Yes	No
ThermistorB	For subsystems that support thermistor measurements, specify the B coefficient for the thermistor that is connected to a particular analog input channel. This coefficient is used by the Steinhart-Hart transfer function.	Yes	No
ThermistorC	For subsystems that support thermistor measurements, specify the C coefficient for the thermistor that is connected to a particular analog input channel. This coefficient is used by the Steinhart-Hart transfer function.	Yes	No

Table 2: Adaptor-Specific Channel Properties (cont.)

Property	Description	A/D	D/A
StrainBridgeConfiguration	For subsystems that support strain gage inputs, specifies the strain gage configuration for a particular analog input channel. The following strain gage configurations are supported: FullBridgeBending, FullBridgeBendingPoisson, FullBridgeAxial, HalfBridgePoisson, HalfBridgeBending, QuarterBridge, and QuarterBridgeTempCompensation.	Yes	No
BridgeConfiguration	For subsystems that support bridge-based inputs, specifies the bridge configuration (FullBridge, HalfBridge, or QuarterBridge) for a particular analog input channel.	Yes	No
StrainShuntResistorEnabled	For devices that support strain gage inputs, specifies whether the shunt resistor is enabled or disabled for a particular analog input channel. If this value is Yes, the shunt resistor is enabled. If this value is No, the shunt resistor is disabled.	Yes	No

a. This property is not supported for the DT9805 and DT9806 modules; customers of these modules must use their own algorithms to convert voltage to temperature based on the thermocouple type they are using. This property is supported by MEASURpoint instruments as well as the DT9828 and DT9828E.

Configuring a Channel for a Voltage Input

If your subsystem supports multiple sensor inputs for each channel, configure the channel for a voltage input using the DTOL-specific **MultiSensorType** property. For example, the following command configures the first channel (channel 0), which is associated with analog input subsystem (ai0), for a voltage measurement:

```
ai0.channel(1).MultiSensorType ='VoltageIn'
```

Some voltage input channels support a bias return termination resistor. The bias return termination resistor is typically enabled for floating and grounded voltage sources, and is typically disabled for voltage sources with grounded references. Refer to the documentation for your device for wiring information.

You can enable or disable the bias return termination resistor for a given channel using the **InputTerminationEnabled** property.

The following example enables the bias return termination resistor for analog input channel 0:

```
ai0.channel(1).InputTerminationEnabled ='Yes'
```

Configuring a Channel for a Current Measurement

If your subsystem supports multiple sensor inputs for each channel, configure the channel for a current measurement using the DTOL-specific **MultiSensorType** property. For example, the following command configures the first channel (channel 0), which is associated with analog input subsystem (ai0), for a current measurement:

```
ai0.channel(1).MultiSensorType ='Current'
```

Some current input channels support a bias return termination resistor. The bias return termination resistor is typically enabled for floating and grounded current sources, and is typically disabled for current sources with grounded references. Refer to the documentation for your device for wiring information.

You can enable or disable the bias return termination resistor for a given channel using the **InputTerminationEnabled** property.

The following example disables the bias return termination resistor for analog input channel 0:

```
ai0.channel(1).InputTerminationEnabled ='No'
```

Configuring a Channel for an IEPE (Accelerometer) Input

If your subsystem supports multiple sensor inputs for each channel, configure the channel for an IEPE (accelerometer) input using the DTOL-specific **MultiSensorType** property. For example, the following command configures the first channel (channel 0), which is associated with analog input subsystem (ai0), for an accelerometer sensor type:

```
ai0.channel(1).MultiSensorType ='Accelerometer'
```

Then, use the properties **Coupling** and **ExcitationCurrentSource** to the configure the coupling type and excitation current source for the channel. Values for the coupling type can be AC or DC. Values for the excitation current source can be Internal, External, or Disabled. If you specify an Internal excitation current source, specify the value of the current source (as a floating-point number) using the **ExcitationCurrentValue** property.

The following example configures the same analog input channel for an internal excitation current source of 4 mA and AC coupling:

```
ai0.channel(1).Coupling='AC'
ai0.channel(1).ExcitationCurrentSource='Internal'
ai0.channel(1).ExcitationCurrentValue= 0.004
```

To configure the same channel for an external excitation current source and DC coupling, use these commands:

```
ai0.channel(1).ExcitationCurrentSource='External'
ai0.channel(1).Coupling='DC'
```

To disable the excitation current source for the same channel, use this command:

```
ai0.channel(1).ExcitationCurrentSource='Disabled'
```

Configuring a Channel for a Thermocouple Input

If your subsystem supports multiple sensor inputs for each channel, configure the channel for a thermocouple input using the DTOL-specific **MultiSensorType** property. For example, the following command configures the first channel (channel 0), which is associated with analog input subsystem (ai0), for a thermocouple measurement:

```
ai0.channel(1).MultiSensorType ='Thermocouple'
```

Then, use the DTOL-specific property **ThermocoupleType** to the configure the type of thermocouple that is connected to the channel. The following values are supported: J, K, B, E, N, R, S, T, or None.

If you specify 'None' for **ThermocoupleType**, data is read as voltage from the corresponding channel and the **UnitRange** reflects the voltage range that is supported by the device.

If you specify a value other than 'None' for **ThermocoupleType**, data is read as temperature from the corresponding channel and the UnitRange reflects the temperature range, in degrees C, of the specified thermocouple type.

For example, the following command configures the same channel for a J type thermocouple input; the **UnitRange** property for that channel will then reflect the temperature range, in degrees C, of the J type thermocouple input:

```
ai0.channel(1).ThermocoupleType= 'J'
```

Data read using **Getdata** and **Getsample** is returned in floating-point format.

Configuring a Channel for an RTD Input

RTD support is implemented differently for MEASURpoint instruments, which use an IVI-COM driver, and the DT9829 and other devices, which use a DT-Open Layers device driver. The following sections describe how RTDs are supported for these devices.

RTD Support for MEASURpoint Instruments

If you are using a MEASURpoint instrument, which uses an IVI-COM device driver, the following values are supported for **RtdType**:

- PT100_385 – Platinum 100 Ω RTD with European alpha curve of 0.00385
- PT500_385 – Platinum 500 Ω RTD with European alpha curve of 0.00385
- PT1000_385 – Platinum 1000 Ω RTD with European alpha curve of 0.00385
- PT100_392 – Platinum 100 Ω RTD with American alpha curve of 0.00392
- PT500_ 392 – Platinum 500 Ω RTD with American alpha curve of 0.00392
- PT1000_392 – Platinum 1000 Ω RTD with American alpha curve of 0.00392
- Volts (if you want to measure voltage instead)
- Ohms (if you want to measure resistance instead)

For example, the following command configures analog input channel 0 for a Platinum 1000 Ω RTD with American alpha curve of 0.00392:

```
ai0.channel(1).RtdType = 'PT1000_392'
```

The **UnitRange** property for that channel will then reflect the temperature range, in degrees C, of the RTD input.

Data read using **Getdata** and **Getsample** is returned in floating-point format.

RTD Support for the DT9829 and Other DT-Open Layers Devices

If you are using a DT9829 module or any other device that uses a DT-Open Layers device driver, and your subsystem supports multiple sensor inputs for each channel, configure the channel for an RTD input using the DTOL-specific **MultiSensorType** property. For example, the following command configures the first channel (channel 0), which is associated with analog input subsystem (ai0), for an RTD measurement:

```
ai0.channel(1).MultiSensorType ='Rtd'
```

Then, use the DTOL-specific property **RtdType** to the configure the type of RTD that is connected to the channel. DT-Open Layers device drivers support the following values for **RtdType**:

- Pt3750
- Pt3850
- Pt3911
- Pt3916
- Pt3920
- Pt3928
- Custom

RtdType identifies the Temperature Coefficient of Resistance (TCR) value that is used by the Callendar-Van Dusen transfer function to determine temperature for RTDs. The Callendar-Van Dusen transfer function is described as follows:

$$R_T = R_0[1 + AT + BT^2 + CT^3(T - 100)]$$

where,

- R_T is the resistance at temperature (TCR).
- R_0 is the resistance at 0° C.
- A, B, and C are the Callendar-Van Dusen coefficients for a particular RTD type. (The value of C is 0 for temperatures above 0° C.)

Table 3 lists the coefficients that are used by the Callendar-Van Dusen transfer function for each **RtdType**.

Table 3: Callendar-Van Dusen Coefficients Supported By RTD Channels

RtdType	Temperature Coefficient of Resistance (TCR)	R0 Coefficient	A Coefficient	B Coefficient	C Coefficient	Applicable Standards
PT3750	0.003750 Ω / Ω /° C	1000 Ω	3.81×10^{-3}	-6.02×10^{-7}	-6.0×10^{-12}	Low Cost
PT3850	0.003850 Ω / Ω /° C	100 Ω, 500 Ω, 1000 Ω	3.9083×10^{-3}	-5.775×10^{-7}	-4.183×10^{-12}	DIN/IEC 60751 ASTM-E1137
PT3911	0.003911 Ω / Ω / ° C	100 Ω	3.9692×10^{-3}	-5.8495×10^{-7}	-4.233×10^{-12}	US Industrial Standard
PT3916	0.003916 Ω / Ω /° C	100 Ω	3.9739×10^{-3}	-5.870×10^{-7}	-4.4×10^{-12}	Japanese JISC 1604-1989
PT3920	0.003920 Ω / Ω /° C	98.129 Ω	3.9787×10^{-3}	-5.869×10^{-7}	-4.167×10^{-12}	SAMA RC21-4-1966
PT3928	0.003928 Ω / Ω /° C	100 Ω	3.9888×10^{-3}	-5.915×10^{-7}	-3.85×10^{-12}	ITS-90

If you specify a value of Pt3850 for the **RtdType**, you must also specify the R0 value using the **RtdR0** property, unless you are using a 100 Ω RTD (the default value). If you specify a value of Custom for the **RtdType**, use the **RtdR0**, **RtdA**, **RtdB**, and **RtdC** properties to specify the values of the R0, A, B, and C coefficients that are used in the Callander Van-Dusen transfer function. For all other **RtdTypes**, the software automatically sets the appropriate value for R0, A, B, and C coefficients based on the selected **RtdType**.

Use the **SensorWiringConfiguration** property to specify the wiring configuration (TwoWire, ThreeWire, or FourWire) for the RTD input. Ensure that the software configuration matches the hardware configuration.

For example, the following commands configure analog input channel 0 for a PT3850 **RtdType** that uses a four-wire configuration:

```
ai0.channel(1).RtdType = 'PT3850'
ai0.channel(1).RtdR0 = 1000
ai0.channel(1).RtdA = 0.0039083
ai0.channel(1).RtdB = -5.775E-07
ai0.channel(1).RtdC = -4.183E-12
ai0.channel(1).SensorWiringConfiguration = 'FourWire'
```

The **UnitRange** property for that channel reflects the temperature range, in degrees C, of the RTD input.

Configuring a Channel for a Thermistor Input

If your subsystem supports multiple sensor inputs for each channel, configure the channel for a thermistor input using the DTOL-specific **MultiSensorType** property. For example, the following command configures the first channel (channel 0), which is associated with analog input subsystem (ai0), for a thermistor measurement:

```
ai0.channel(1).MultiSensorType ='Thermistor'
```

To determine the temperature of a thermistor, the software uses the Steinhart-Hart equation:

$$\frac{1}{T} = A + BInR + CIn(R)^3$$

where,

- T is the temperature.
- R is the resistance at T, in ohms, that is supplied by the device.
- A, B, and C are the Steinhart-Hart coefficients for a particular thermistor type and value, and are supplied by the thermistor manufacturer.

Use the **ThermistorA**, **ThermistorB**, and **ThermistorC** properties to specify the A, B, and C coefficients used in the Steinhart-Hart transfer function.

Use the **SensorWiringConfiguration** property to specify the wiring configuration (TwoWire, ThreeWire, or FourWire) for the thermistor input. Ensure that the software configuration matches the hardware configuration.

For example, the following commands configure analog input channel 0 for a thermistor measurement that uses a two-wire configuration:

```
ai0.channel(1).ThermistorA = 0.001032
ai0.channel(1).ThermistorB = 0.0002387
ai0.channel(1).ThermistorC = 1.58E-07
ai0.channel(1).SensorWiringConfiguration = 'TwoWire'
```

The **UnitRange** property for that channel reflects the temperature range, in degrees C, of the thermistor input.

Configuring a Channel for a Resistance Measurement

Note: To measure resistance on MEASURpoint instruments, specify the channel for an RTD measurement, as described on page 45, and choose Ohms as the **RtdType**.

If your subsystem supports multiple sensor inputs for each channel, configure the channel for a resistance measurement using the DTOL-specific **MultiSensorType** property. For example, the following command configures the first channel (channel 0), which is associated with analog input subsystem (ai0), for a resistance measurement:

```
ai0.channel(1).MultiSensorType ='Resistance'
```

Then, use the **ExcitationCurrentSource** property to the configure the excitation current source for the channel. Values for the excitation current source can be Internal, External, or Disabled. If you specify an Internal excitation current source, specify the value of the current source (as a floating-point number) using the **ExcitationCurrentValue** property.

Use the **SensorWiringConfiguration** property to specify the wiring configuration (TwoWire, ThreeWire, or FourWire) for the resistor that is connected to the channel. Ensure that the software configuration matches the hardware configuration.

The following example configures analog input channel 0 for an internal excitation current source of 425 µA and a three-wire configuration:

```
ai0.channel(1).ExcitationCurrentSource ='Internal'
ai0.channel(1).ExcitationCurrentValue = 0.000425
ai0.channel(1).SensorWiringConfiguration = 'ThreeWire'
```

Configuring a Channel for a Strain Gage Input

If your subsystem supports multiple sensor inputs for each channel, configure the channel for a strain gage measurement using the DTOL-specific **MultiSensorType** property. For example, the following command configures the first channel (channel 0), which is associated with analog input subsystem (ai0), for a strain gage measurement:

```
ai0.channel(1).MultiSensorType ='StrainGage'
```

Use the DTOL-specific properties **StrainBridgeConfiguration** and **StrainShuntResistorEnabled** to the configure a channel for a strain gage input. Possible values for **StrainBridgeConfiguration** are as follows: FullBridgeBending, FullBridgeBendingPoisson, FullBridgeAxial, HalfBridgePoisson, HalfBridgeBending, QuarterBridge, and QuarterBridgeTempCompensation. Possible values for **StrainShuntResistorEnabled** are Yes (to enable the shunt resistor) and No (to disable the shunt resistor.

For example, the following commands configure the first channel (channel 0), which is associated with the analog input subsystem (ai0), for a strain gage input that uses a Full Bridge Bending bridge configuration with the shunt resistor enabled:

```
ai0.channel(1).StrainBridgeConfiguration='FullBridgeBending'
ai0.channel(1).StrainShuntResistorEnabled='Yes'
```

Configuring a Channel for a Bridge-Based Sensor

If your subsystem supports multiple sensor inputs for each channel, configure the channel for measuring a bridge-based sensor or general-purpose bridge using the DTOL-specific **MultiSensorType** property. For example, the following command configures the first channel (channel 0), which is associated with analog input subsystem (ai0), for a bridge-based sensor measurement:

```
ai0.channel(1).MultiSensorType ='Bridge'
```

Use the DTOL-specific properties **BridgeConfiguration** and **StrainShuntResistorEnabled** to the configure a channel for a bridge-based sensor or general-purpose bridge input. Possible values for **BridgeConfiguration** are FullBridge, HalfBridge, and QuarterBridge. Possible values for **StrainShuntResistorEnabled** are Yes (to enable the shunt resistor) and No (to disable the shunt resistor.

For example, the following commands configure the first channel (channel 0), which is associated with the analog input subsystem (ai0), for a bridge-based sensor with the shunt resistor disabled:

```
ai0.channel(1).BridgeConfiguration='FullBridge'
ai0.channel(1).StrainShuntResistorEnabled='No'
```

Configuring the Gain of a Channel

Use the DTOL-specific **GainPerChan** property to set the gain for a channel. For example, this command sets the gain for the first channel (channel 0) of the analog input subsystem (ai0) to 4:

```
ai0.channel(1).GainPerChan = 4
```

Note: DT-Open Layers uses a *per-channel gain* concept, which differs from MATLAB's *range based per channel* setting to control gain and range. You can specify a per-channel gain, or the adaptor attempts to set the best per-channel gain for a specified range.

We recommend using the **GainPerChan** property for per-channel gain control.

Utility Methods in MATLAB

A MATLAB file called DtStrainGageUtils.m is provided for users who want to convert voltage data from a strain gage input into strain gage units or voltage data from a bridge-based sensor into the units of a bridge-based sensor.

DtStrainGageUtils.m contains the following utility methods:

Table 4: Utility Methods in DtStrainGageUtils.m for Converting Voltage from Strain Gage and Bridge-Based Sensors in MATLAB

Utility Method	Description
olDaReadBridgeSensorVirtualTeds	If your bridge-based sensor or transducer supports a TEDS (Transducer Electronic Data Sheet) interface, this method allows you to read the TEDS data from a data file (virtual TEDS).
olDaReadStrainGageVirtualTeds	If your strain gage input supports a TEDS (Transducer Electronic Data Sheet) interface, this method allows you to read the TEDS data from a data file (virtual TEDS).
VoltstoMicroStrain	Converts a raw A/D count value into a strain value based on a specified configuration.
VoltstoBridgeBasedSensor	Converts a raw A/D count value into a value for a bridge-based sensor based on a specified configuration.

olDaReadBridgeSensorVirtualTeds

If your bridge-based sensor or transducer supports a TEDS (Transducer Electronic Data Sheet) interface, this method allows you to read the TEDS data from a data file (virtual TEDS).

The format of this method is as follows:

```
TedsBridgeData = olDaReadBridgeSensorVirtualTeds (
   virtualTedsFileName)
```

where,

- *virtualTedsFileName* is the full path, including the file name of the TEDS data file.
- *TedsBridgeData* is a structure of type BRIDGE_SENSOR_TEDS_tag containing the TEDS data for the bridge-based sensor. This structure has the following elements; refer to the file OldaapiExports.m for more information on this structure:

Table 5: Elements of the BRIDGE_SENSOR_TEDS_tag Structure

Element	Description
manufacturerId	Part of the Basic TEDS information, identifies the manufacturer of the sensor for the channel.
modelNumber	Part of the Basic TEDS information, identifies the model number of the sensor for the channel.
versionLetter	Part of the Basic TEDS information, identifies the version letter of the sensor for the channel.
versionNumber	Part of the Basic TEDS information, identifies the version number of the sensor for the channel.
serialNumber	Part of the Basic TEDS information, identifies the serial number of the sensor for the channel.
calDaysSince1_1_1998	The calibration date that was specified in the TEDS data for the channel.
calInitials	The calibration initials that were specified in the TEDS data for the channel.
calPeriod	The calibration period that was specified in the TEDS data for the channel.
exciteAmplMax	The maximum excitation voltage that was specified in the TEDS data for the channel
exciteAmplMin	The minimum excitation voltage that was specified in the TEDS data for the channel.
exciteAmplNom	The nominal excitation voltage that was specified in the TEDS data for the channel.
maxElecVal	The maximum electrical output, in V/V, that was specified in the TEDS data for the channel.
maxPhysicalValue	The positive full-scale value, in strain, that was specified in the TEDS data for the channel.
measID	The measurement location ID that was specified in the TEDS data for the channel.
minElecVal	The minimum electrical output, in V/V, that was specified in the TEDS data for the channel.
minPhysicalValue	The negative full-scale value, in strain, that was specified in the TEDS data for the channel.
physicalMeasurand	The physical Measurand units that were specified in the TEDS data for the channel. Possible values are as follows: - Temperature_Kelvin - Temperature (Kelvin) - Temperature_Celsius - Temperature (Celsius) - Strain - Strain - Microstrain - Microstrain - Newton - Force/Weight (Newton) - pounds - Force/Weight (pounds) - kilogram ForcePer Kilopound - Force/Weight (kilogram-force/kilopound) - Acceleration_m_ss - Acceleration (m/s^2) - Acceleration_g - Acceleration (g) - Torque_Nm_Radian - Torque (Nm/radian) - Torque_Nm - Torque (Nm) - Torque_oz_in - Torque (oz-in) - Pressure_Pascal - Pressure (Pascal) - Pressure_PSI - Pressure (PSI) - Mass_Kg - Mass (kg) - Mass_g - Mass (g) - Distance_m - Distance (m) - Distance_mm - Distance (mm) - Distance_inches - Distance (inches)

Table 5: Elements of the BRIDGE_SENSOR_TEDS_tag Structure (cont.)

Element	Description
physicalMeasurand (cont.)	- Velocity_m_s - Velocity (m/s) - Velocity_mph - Velocity (mph) - Velocity_fps - Velocity (fps) - AngularPosition_radian- Angular Position (radian) - AngularPosition_degrees - Angular Position (degrees) - RotationalVelocity_radian_s - Rotational Velocity (radian/s) - RotationalVelocity_rpm - Rotational Velocity (rpm) - Frequency - Frequency - Concentration_gram_liter - Concentration (gram/liter) - Concentration_kg_liter - Concentration (kg/liter) - MolarConcentration_mole_m3 - Molar Concentration (mole/m^3) - MolarConcentration_mole_l - Molar Concentration (mole/l) - VolumetricConcentration_m3_m3 - Volumetric Concentration (m^3/m^3) - VolumetricConcentration_l_l - Volumetric Concentration (l/l) - MassFlow - Mass Flow - VolumetricFlow_m3_s - Volumetric Flow (m^3/s) - VolumetricFlow_m3_hr - Volumetric Flow (m^3/hr) - VolumetricFlow_gpm - Volumetric Flow (gpm) - VolumetricFlow_cfm - Volumetric Flow (cfm) - VolumetricFlow_l_min - Volumetric Flow (l/min) - RelativeHumidity - Relative Humidity - Ratio_percent - Ratio (percent) - Voltage - Voltage - RmsVoltage - RMS Voltage - Current - Current - RmsCurrent - RMS Current - Power_Watts - Power (Watts)
respTime	The response time, in seconds, that was specified in the TEDS data for the channel.
selector	The full-scale electrical value precision (minimum voltage output for 11-, 19-, or 32-bits; or maximum voltage output for 11-, 19-, or 32-bits) that was specified in the TEDS data for the channel.
sensorImped	The bridge element impedance, in ohms, that was specified in the TEDS data for the channel.
tedsBridgeType	The type of bridge (Full Bridge, Half Bridge, or Quarter Bridge) that was specified in the TEDS data for the channel.

olDaReadStrainGageVirtualTeds

If your strain gage input supports a TEDS interface, this method allows you to read the TEDS data from a data file (virtual TEDS).

The format of this method is as follows:

```
TedsStrainData = olDaReadStrainGageVirtualTeds (virtualTedsFileName)
```

where,

- *virtualTedsFileName* is the full path, including the file name of the TEDS data file.
- *TedsStrainData* is a structure of type BSTRAIN_GAGE_TEDS_tag containing the TEDS data for the strain gage. This structure has the following elements; refer to the file OldaapiExports.m for more information on this structure:

Table 6: Elements of the STRAIN_GAGE_TEDS_tag Structure

Element	Description
manufacturerId	Part of the Basic TEDS information, identifies the manufacturer of the sensor for the channel.
modelNumber	Part of the Basic TEDS information, identifies the model number of the sensor for the channel.
versionLetter	Part of the Basic TEDS information, identifies the version letter of the sensor for the channel.
versionNumber	Part of the Basic TEDS information, identifies the version number of the sensor for the channel.
serialNumber	Part of the Basic TEDS information, identifies the serial number of the sensor for the channel.
minPhysicalValue	The negative full-scale value, in strain, that was specified in the TEDS data for the channel.
maxPhysicalValue	The positive full-scale value, in strain, that was specified in the TEDS data for the channel.
minElecVal	The minimum electrical output, in V/V, that was specified in the TEDS data for the channel.
maxElecVal	The maximum electrical output, in V/V, that was specified in the TEDS data for the channel.
gageType	The type of gage that was specified in the TEDS data for the channel. Possible values are as follows: - SingleElement - Single element gage. - TwoPoissonElements - Two elements with a Poisson arrangement. - TwoOppositeSignedElements - Two elements, opposite sign (adjacent arms). - TwoSameSignedElements - Two elements, same sign (opposite arms). - TwoElementChevron - Two elements, 45° Chevron (torque or shear) arrangement. - FourSameSignElementsPoisson - Four elements, Poisson strains of same sign in opposite arms. - FourOppositeSignedElements - Four elements, Poisson strains of opposite sign in adjacent arms. - FourUniaxialElements - Four elements, equal strains of opposite sign in adjacent arms. - FourElementDualChevron - Four elements, 45° Chevron (torque or shear) arrangement. - TeeRosetteGrid1_0Degrees - Tee Rosette grid 1 or a (0°). - TeeRosetteGrid2_90Degrees - Tee Rosette grid 2 or b (90°). - DeltaRosetteGrid1_0Degrees - Delta Rosette grid 1 or a (0°). - DeltaRosetteGrid2_60Degrees - Delta Rosette grid 2 or b (60°). - DeltaRosetteGrid3_120Degrees - Delta Rosette grid 3 or c (120°). - RectangularRosetteGrid1_0Degrees - Rectangular Rosette grid 1 or a (0°). - RectangularRosetteGrid2_45Degrees - Rectangular Rosette grid 2 or a (45°). - RectangularRosetteGrid3_90Degrees - Rectangular Rosette grid 3 or a (90°).
gageFactor	The gage factor, or sensitivity of the strain gage, that was specified in the TEDS data for the channel.
gageTransSens	The transverse sensitivity, in percentage, that was specified in the TEDS data for the channel.
gageOffset	The zero offset value after installation, in V/V, that was specified in the TEDS data for the channel.
poissonCoef	The Poisson coefficient after installation that was specified in the TEDS data for the channel.
youngsMod	The Young's modulus, or measure of the stiffness of the material, in MPa, that was specified in the TEDS data for the channel.

Table 6: Elements of the STRAIN_GAGE_TEDS_tag Structure (cont.)

Element	Description
gageArea	The area of each gage element, in mm², that was specified in the TEDS data for the channel.
tedsBridgeType	The type of bridge (Quarter, Half, or Full) hat was specified for the channel.
sensorImped	The bridge element resistance, in ohms, that was specified in the TEDS data for the channel.
respTime	The response time, in seconds, that was specified in the TEDS data for the channel.
exciteAmplNom	The nominal excitation voltage that was specified in the TEDS data for the channel.
exciteAmplMax	The maximum excitation voltage that was specified in the TEDS data for the channel.
calDaysSince1_1_1998	The calibration date that was specified in the TEDS data for the channel.
calInitials	The calibration initials that were specified in the TEDS data for the channel.
calPeriod	The calibration period that was specified in the TEDS data for the channel.
measID	The measurement location ID that was specified in the TEDS data for the channel.

VoltstoMicroStrain

This method converts a raw A/D count value into a strain value based on a specified configuration.

The format of this method is as follows:

```
MicroStrainResults = VoltstoMicroStrain (BridgeConfig, Vu, Vs, Vex,
   GF, Rg, Rl, Pr, ShuntCorrection)
```

The arguments to the method are as follows:

Table 7: Arguments to VoltstoMicroStrain

Argument	Description
BridgeConfig	The configuration of the bridge. Values are as follows: FULL_BRIDGE_BENDING, FULL_BRIDGE_BENDING_POISSON, FULL_BRIDGE_AXIAL, HALF_BRIDGE_POISSON, HALF_BRIDGE_BENDING, QUARTER_BRIDGE, QUARTER_BRIDGETEMPCOMPENSATION
Vu	The initial value of the bridge output (in voltage) in the unstrained/unloaded condition.
Vs	The measured voltage output of the bridge in the strained condition.
Vex	The excitation voltage.
GF	The gage factor, or sensitivity, of the bridge. Refer to documentation from the manufacturer of the strain gage.
Rg	The nominal gage resistance of the bridge, in ohms. Refer to documentation from the manufacturer of the strain gage.

Table 7: Arguments to VoltstoMicroStrain (cont.)

Argument	Description
Rl	The lead wire resistance, in ohms. Specify 0 for this parameter if remote sensing is used. Remote sensing ensures the correct Vex at the bridge so any wire resistance is effectively negated.
Pr	The Poisson ratio for the bridge, defined as the negative ratio of transverse strain to axial (longitudinal) strain.
ShuntCorrection	The shunt correction value.

VoltstoBridgeBasedSensor

This method converts a raw A/D count value into a value for a bridge-based sensor based on a specified configuration.

The format of this method is as follows:

```
BridgeSensorResults = VoltstoBridgeBasedSensor (Vu, Vs, Vex, Tc, Rg,
  Rl, RolnmV_V, ShuntCorrection)
```

The arguments to the method are as follows:

Table 8: Arguments to VoltstoBridgeBasedSensor

Argument	Description
Vu	The initial value of the bridge output (in voltage) in the unstrained/unloaded condition.
Vs	The measured voltage output of the bridge in the strained condition.
Vex	The excitation voltage.
Tc	The full-scale range of the transducer in its native engineering units.
Rg	The nominal gage resistance of the bridge, in ohms.
Rl	The lead wire resistance, in ohms. Specify 0 for this parameter if remote sensing is used. Remote sensing ensures the correct Vex at the bridge so any wire resistance is effectively negated.
RoInmv_V	The rated output of the transducer in terms of mV/V excitation.
ShuntCorrection	The shunt correction value.

MATLAB Property Notes

The following adaptor-specific notes apply to MATLAB properties:

Table 9: Adaptor Notes for MATLAB Properties

MATLAB Property	Notes
BufferingConfig	The minimum buffer size is 128, but for best performance, set to 1024 or greater. **NOTE:** If the value for **RepeatOutput** is greater than 1, the buffer size must be an integer multiple of the number of samples queued. The adaptor warns you of this when you change RepeatOutput.
TotalChannels	This may include channels that are in two lists and have a mode (single-ended vs. differential). Also, when the InputType is changed from SingleEnded to Differential, the channel IDs will change in the DaqHwInfo() listing.
TriggerCondition	Either "RisingEdge" or "FallingEdge" depending upon what the module supports.

Multiple Subsystems of a Single Type

To provide access to all elements of a device with multiple subsystems of a particular type (for example, multiple analog input subsystems, such as on the DT9822), the adaptor creates *virtual boards*. The first board enumerated includes any other types of subsystems.

For example, the DT9822 module enumerates in the adaptor like this:

'DT9822(01)'	A/D, D/A, Digital I/O
'DT9822(01)-1'	A/D
'DT9822(01)-2'	A/D
'DT9822(01)-3'	A/D

To access the fourth A/D subsystem, use the virtual board 'DT9822(01)-3' and its one analoginput() object.

Analog Output Operation Notes

The following caveats apply to analog output operations:

- Before starting an analog output operation, use the **OutofDataMode** command if you want to reset the output to 0 V after the buffer has completed. An example follows:

  ```
  set (a0, 'OutofDataMode', 'DefaultValue');
  ```

- Whenever you want to perform analog output streaming, it is recommended that you add the following two lines of code during the configuration process and before starting the analog output operation:

  ```
  set (a0, 'BufferingMode', 'manual');
  set (a0, 'BufferingConfig',[samplesPerBuffer,10]);
  ```

  These two lines improve performance as the engine buffers used in MATLAB will match the DT-Open Layers buffers and the buffers passed in the **PutData** method.

  The first line allows MATLAB to set the buffer size of the engine to correspond to the size of the buffer used in **PutData**.

  In the second line, *samplesPerBuffer* is the number of samples in each waveform used in **PutData**.

- The number of samples output will not always be correct, particularly when the samples to output are not the same as the buffer size. This is because DT-Open Layers sends the whole buffer to the D/A converter, and the adaptor pads the output buffer to either the Hold value or the channel default value (see **OutofDataMode**).

- DT-Open Layers does not send a message when the actual hardware trigger occurs. Therefore, the time and sample count for the trigger event will not be correct for **TriggerType='HwDigital'** for analog out.

- The final voltage on a output channel might be unpredictable after a streaming operation.

Using Hardware Triggers

To use a hardware trigger for analog output operations, set the **TriggerType** to 'HwDigital' and set the **TriggerCondition** property to Rising or Falling.

When the analog output subsystem is started, data buffers may be sent to the hardware FIFO, resulting in a "buffer done" message to the adaptor and a trigger event before the D/A output value has been set. Therefore it may appear that data output occurs before the hardware trigger happens, but the data has only been moved to the board FIFO, not output through the D/A converter.

Index

I

L

M

N

O

R

S

T

U

V

W