DATA TRANSLATION

UM-22024-L

DAQ Adaptor for MATLAB®

Eleventh Edition
February, 2014

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

Overview

This document describes how to install and use the Data Translation DAQ Adaptor for MATLAB®.

Data Translation's DAQ Adaptor for MATLAB provides an interface between the MATLAB Data Acquisition (DAQ) subsystem from The MathWorks and Data Translation's DT-Open Layers architecture.

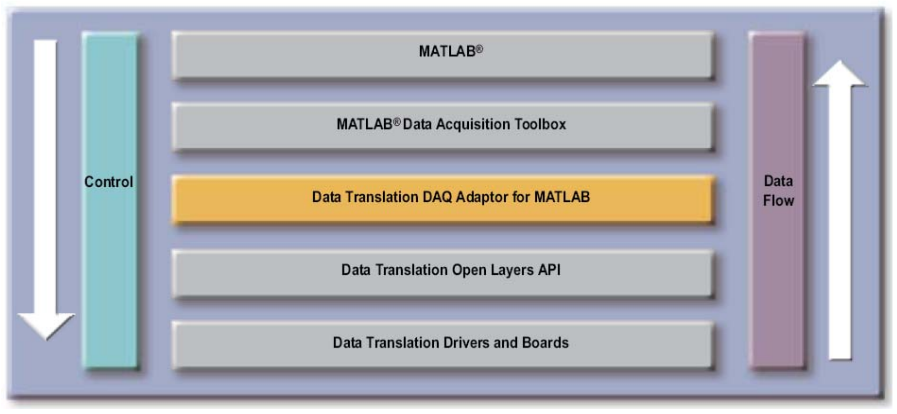
DT-Open Layers provides an API to all of Data Translation's analog and digital I/O data acquisition hardware modules. You can use these two components together to control Data Translation modules with MATLAB and move data directly into MATLAB for analysis and display.

Note: Counter/timer operations are not supported in MATLAB unless they are implemented as part of the analog I/O data stream.

Using these components together, you can create DT-Open Layers subsystems, add channels to them, and perform single-value or streaming I/O through them to MATLAB. MATLAB provides extensive analysis and display features to process the input data and generate output data.

Data Flow Model

The figure below summarizes the relationship between the MATLAB and Data Translation components in a combined configuration:


Control
MATLAB®
MATLAB® Data Acquisition Toolbox
Data Translation DAQ Adaptor for MATLAB
Data Translation Open Layers API
Data Translation Drivers and Boards
Data Flow

Requirements

You must install the following before you install the 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)
- MATLAB Version 7 (R14) Service Pack 3 or higher
- MATLAB Data Acquisition Toolbox Version 2.7 or higher

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 DAQ Adaptor for MATLAB, do the following:

1. Click the link for the adaptor on Data Translation's web site.
2. Choose **Open** from the Windows dialog box, or choose **Save** to download the file before proceeding. If you chose the save the file, double-click **setup.exe** after you download it.
3. Close other programs if necessary, and click **Next** on the welcome screen.
4. 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.
5. Choose the **Yes** or **No** radio button, and then click **Next.**
   The installer prompts you to begin file installation.
6. Click **Next.**
   The installer copies the files to the destination directory.

   After the installer copies the files, it 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
   >>
   ```
7. When you return to the installer, click **Finish.**

Using the DAQ Adaptor for MATLAB

The Data Translation 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 MATLAB 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.TempFilterType='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 15, 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 olDaReadStrainGageVirtualTeds

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 Out 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 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.

Related Information

Refer to the following documents from Data Translation for more information on using DT-Open Layers:

- *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.

Index