

## **IVI Class Drivers Overview**

---

The Measurement Studio IVI class driver classes provide native Measurement Studio interfaces to IVI class-compliant instrument drivers to allow you to seamlessly integrate instrument control into your Measurement Studio applications. You can use the Measurement Studio IVI class drivers to control any instrument that has an IVI class-compliant instrument driver.

### **Top-Level Classes**

---

[CNiIviDcPwr](#) - CNiIviDcPwr controls DC power supplies that have IVI class-compliant instrument drivers.

[CNiIviDmm](#) - CNiIviDmm controls digital multimeters (DMMs) that have IVI class-compliant instrument drivers.

[CNiIviFgen](#) - CNiIviFgen controls function generators that have IVI class-compliant instrument drivers.

[CNiIviScope](#) - CNiIviScope controls oscilloscopes that have IVI class-compliant instrument drivers.

[CNiIviSwitch](#) - CNiIviSwth controls switches that have IVI class-compliant instrument drivers.

### **Exceptions**

---

[CNiIviException](#) – All instrument driver classes throw an instance of this class when an error occurs.

# Measurement Studio IVI DC Power Driver Overview

---

You use CNiIviDcPwr to control DC power supplies that have IVI class compliant instrument drivers.

## Top-Level Classes

---

[CNiIviDcPwr](#)

# SimpleIviDcPwr Example Program

This topic includes a summary of and links to the SimpleIviDcPwr example program. You can load the examples in either Microsoft Visual C++ 6.0 or Microsoft Visual C++ .NET. When you run an example, Measurement Studio Reference determines what version of Visual C++ you have installed and runs the example accordingly.

<b>Example</b>	<b>Description</b>	<b>Visual C++ 6.0</b>	<b>Visual C++ .NET</b>	<b>Run Example</b>
SimpleIviDcPwr	The SimpleIviDcPwr example demonstrates demonstrates how to use the CNiIviDcPwr class and the SamplePowerSupply virtual instrument driver to simulate a power supply.	<a href="#">Load</a>	<a href="#">Load</a>	<a href="#">Run</a>

# **CNiIviClassDriver**



**Class**

Declared in:  
**NiIviClassDriver.h**

## Overview

---

CNiIviClassDriver is the base class from which all class driver classes inherit. CNiIviClassDriver provides functionality common to all class drivers.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

▲ CNiViDriver

## ◆ Data Items

---

- ◆ CNiIviClassDriverIdentity **ClassDriverIdentity** Information about the class driver.
- ◆ CNiIviClassDriverOperation **ClassDriverOperation** Properties that control the behavior of the class driver.

## ◆ **F u n c t i o n s**

---

- ◆ `inline` ViSession [GetSpecificDriverSession\(\)](#) Returns a session to the specific driver in use by the class driver.  
`const`
-



# **CNiIviClassDriverIdentity**

**Class**

Declared in:  
**NiIviClassDriver.h**

## Overview

---

Information about the class driver.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ <u>CString</u>	<u>Description</u>	Description of the class driver.
◆ <u>CString</u>	<u>Identifier</u>	A unique identifier for the class driver.
◆ <u>CString</u>	<u>Revision</u>	The revision information for the class driver.
◆ <u>long</u>	<u>SpecificationMajorVersion</u>	The major version of the Ivi Foundation class specification to which this class driver conforms.
◆ <u>long</u>	<u>SpecificationMinorVersion</u>	The minor version of the Ivi Foundation class specification to which this class driver conforms.
◆ <u>CString</u>	<u>SpecificDriverLocator</u>	The location of the specific driver module in use by the class driver.
◆ <u>CString</u>	<u>Vendor</u>	The class driver vendor.

---

# **CNiIviClassDriverOperation**

**Class**

Declared in:  
**NiIviClassDriver.h**

## **Overview**

---

Properties that control the behavior of the class driver.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ `bool`      Spy      Specifies whether to log class function calls to the NI Spy utility.
  - ◆ `bool`      UseSpecificSimulation      Specifies whether to simulate instrument driver I/O operations in the specific or class driver when simulation is enabled.
-

 **CNiIviDcPwr**



**Class**

Declared in:  
**NiIviDcPwr.h**

## Overview

---

CNiIviDcPwr controls DC power supplies that have IVI class compliant instrument drivers. CNiIviDcPwr encapsulates the functionality of a DC power supply. CNiIviDcPwr allows you to configure the voltage level, current limit, current limit behavior, and over voltage protection limit. CNiIviDcPwr also allows you to enable and disable output channels with a DC power supply. CNiIviDcPwr also allows you to configure the output range in which the power supply operates, and to query the instrument to determine operating status. CNiIviDcPwr supports typical DC power supplies as well as common extended functionality found in more complex instruments.

 [Hierarchy Chart](#)



## ▲ Base Classes

---

▲ CNiIviClassDriver

## ◆ Data Items

---

- ◆ [long](#) **ChannelCount** The number of channels that the specific driver supports.
- ◆ [CNiIviDcPwrOutput](#) **Output** Additional settings for configuring the power supply's output signal.
- ◆ [CNiIviDcPwrTrigger](#) **Trigger** Additional settings for triggering changes in the power signal.

## ◆ Constructors

---

- ◆ `inline CNIviDcPwr()` Default constructor.
- ◆ `inline CNIviDcPwr( const CString &resourceName, bool idQuery = true, bool reset = true, const CString &options = "" )` Constructor.

## ◆ **Functions**

---

◆ `inline CString GetChannelName( ViInt32 index ) const`

---

---

# **CNiIviDcPwrOutput**



**Class**

Declared in:  
**NiIviDcPwr.h**

## Overview

---

Additional settings for configuring the power supply's output signal.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ double	<u>CurrentLimit</u> []	The output current limit in Amperes.
◆ long	<u>CurrentLimitBehavior</u> []	The behavior of the power supply when the output current is equal to or greater than the value of the CurrentLimit property.
◆ bool	<u>Enabled</u> []	Specifies whether the signal the power supply produces appears at the output connector.
◆ bool	<u>OvpEnabled</u> []	Specifies whether the power supply provides over-voltage protection.
◆ double	<u>OvpLimit</u> []	The maximum voltage the power supply allows when the OvpEnabled property is true.
◆ double	<u>TriggeredCurrentLimit</u> []	The value in Amperes to which the power supply sets the current limit after a trigger event occurs.
◆ double	<u>TriggeredVoltageLevel</u> []	The value in Volts to which the power supply sets the voltage level after a trigger event occurs.
◆ double	<u>VoltageLevel</u> []	The voltage level in Volts the power supply attempts to generate.

## ◆ Functions

---

- ◆ inline void **ConfigureCurrentLimit**( `const CString&` channelName, `long` behavior, `double` limit ) Configures properties that affect the power supply's current limit.
- ◆ inline void **ConfigureOutputRange**( `const CString` &channelName, `long` rangeType, `double` range ) Configures the power supply's output range on a channel.
- ◆ inline void **ConfigureOvp**( `const CString` &channelName, `bool` enabled, `double` limit ) Configures the power supply's over-voltage protection.
- ◆ inline double **MeasureCurrent**( `const CString` &channelName ) Returns the actual current in Amperes.
- ◆ inline double **MeasureVoltage**( `const CString` &channelName ) Returns the actual voltage in Volts
- ◆ inline double **QueryCurrentLimitMax**( `const CString` &channelName, `double` voltageLevel ) Returns the maximum programmable current limit that the power supply accepts for a particular voltage level on a channel for the output range to which the power supply is currently configured.
- ◆ inline bool **QueryOutputState**( `const CString` &channelName, `long` outputState ) Returns whether the power supply is in a particular output state.
- ◆ inline double **QueryVoltageLevelMax**( `const CString` &channelName, `double` currentLimit ) Returns the maximum programmable voltage level that the power supply accepts for a particular current limit on a channel for the output range to which the power supply is currently configured.



◆ inline void **ResetOutputProtection**( Clears all output-protection  
const CString conditions on the power supply.  
&channelName )

---

# **CNiIviDcPwrTrigger**



**Class**

Declared in:  
**NiIviDcPwr.h**

## Overview

---

Additional settings for triggering changes in the power signal.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ long      Source      The trigger source.

## ◆ **F u n c t i o n s**

---

- ◆ inline void **Abort()** Aborts all pending output changes.
  - ◆ inline void **Initiate()** Initiates previously specified output changes.
  - ◆ inline void **SendSoftwareTrigger()** Sends a command to trigger the power supply.
-

# **CNiIviDriver**



**Class**

Declared in:  
**NilviDriver.h**

## Overview

---

CNiIviDriver is the base class from which all instrument driver classes inherit. CNiIviDriver provides functionality common to all instrument drivers.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ <a href="#">bool</a>	<a href="#"><b><u>DestructorClosesSession</u></b></a>	Determines if the session associated with the object is closed when the object is destroyed.
◆ <a href="#">CNIviDriverOperation</a>	<a href="#"><b><u>DriverOperation</u></b></a>	Properties that control the behavior of the specific driver.
◆ <a href="#">CNIviSpecificDriverIdentity</a>	<a href="#"><b><u>SpecificDriverIdentity</u></b></a>	Information about the specific driver.
◆ <a href="#">CNIviUtility</a>	<a href="#"><b><u>Utility</u></b></a>	Additional utility settings.



## ◆ Functions

---

- ◆ `inline void`      `CloseSession()`      Closes the driver session associated with the object.
  
  - ◆ `inline ViSession` `GetSession() const`      Return the session associated with the object.
  
  - ◆ `inline bool`      `SessionIsValid() const`      Indicates if the session associated with the object is valid.
  
  - ◆ `inline void`      `SetSession( const CString& rsrcName, bool reset, bool idQuery, const CString& options = "", bool destructorClosesSession = true )`      Constructor that creates a new session from the resource descriptor provided and assigns it to the object.
  
  - ◆ `inline void`      `SetSession( ViSession vi, bool destructorClosesSession = false )`      Sets the object session to the value provided.
-

# **CNiIviDriverOperation**



**Class**

Declared in:  
**NiIviDriver.h**

## Overview

---

Properties that control the behavior of the specific driver.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ <code>bool</code>	<u><b>Cache</b></u>	Specifies whether to cache property values.
◆ <code>CString</code>	<u><b>DriverSetup</b></u>	Returns the driver setup string used to initialize the object.
◆ <code>bool</code>	<u><b>InterchangeCheck</b></u>	Specifies whether to perform interchangeability checking and log interchangeability warnings.
◆ <code>CString</code>	<u><b>IoResourceDescriptor</b></u>	Indicates the resource descriptor the driver uses to identify the physical device.
◆ <code>CString</code>	<u><b>LogicalName</b></u>	String containing the logical name used to initialize the object.
◆ <code>bool</code>	<u><b>QueryInstrumentStatus</b></u>	Specifies if the instrument driver queries the instrument status after each operation.
◆ <code>bool</code>	<u><b>RangeCheck</b></u>	Specifies whether to validate parameters passed to instrument driver functions.
◆ <code>bool</code>	<u><b>RecordCoercions</b></u>	Specifies if the instrument driver keeps a list of the value coercions it makes for numerical properties.
◆ <code>bool</code>	<u><b>Simulate</b></u>	Specifies whether to simulate instrument driver I/O operations.

## ◆ Functions

---

- ◆ `inline void`     **ClearInterchangeWarnings()**     Clears the list of current interchange warnings.
  - ◆ `inline cString` **GetNextCoercionRecord()**     Retrieves and clears the oldest instance in which the driver coerced a value you specified to another value.
  - ◆ `inline cString` **GetNextInterchangeWarning()**     Retrieves and clears the oldest instance in which the class driver recorded an interchangeability warning.
  - ◆ `inline void`     **InvalidateAllProperties()**     Invalidates the cached values of all properties.
  - ◆ `inline void`     **ResetInterchangeCheck()**     Causes interchangeability checking algorithms in the specific driver to ignore all previous configuration operations.
-

# **CNiIviException**



**Class**

Declared in:  
**NiIviException.h**

## Overview

---

 [Hierarchy Chart](#)

## ▲ Base Classes

---

▲ CNiExceptionT<ViStatus>



## ◆ **C o n s t r u c t o r s**

---

◆ `inline` [CNilviException](#)( ViStatus code, `const` [CString&](#) msg )

---

---

# **CNiIviSpecificDriverIdentity**

**Class**

Declared in:  
**NiIviDriver.h**

## Overview

---

Information about the specific driver.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ <u>CString</u>	<u>Description</u>	Description of the specific driver.
◆ <u>CString</u>	<u>GroupCapabilities</u>	Comma-delimited list of the extension groups supported by the specific driver.
◆ <u>CString</u>	<u>Identifier</u>	Unique identifier for the specific driver.
◆ <u>CString</u>	<u>InstrumentFirmwareRevision</u>	Instrument firmware revision.
◆ <u>CString</u>	<u>InstrumentManufacturer</u>	Instrument manufacturer.
◆ <u>CString</u>	<u>InstrumentModel</u>	Instrument model.
◆ <u>CString</u>	<u>Revision</u>	Revision information for the specific driver.
◆ <u>long</u>	<u>SpecificationMajorVersion</u>	Major version of the IVI Foundation class specification to which this instrument driver conforms.
◆ <u>long</u>	<u>SpecificationMinorVersion</u>	Minor version of the IVI Foundation class specification to which this instrument driver conforms.
◆ <u>CString</u>	<u>SupportedInstrumentModels</u>	Instrument models supported by the specific driver.
◆ <u>CString</u>	<u>Vendor</u>	Specific driver vendor.

---

# **CNiIviUtility**

**Class**



Declared in:  
**NilviDriver.h**

## Overview

---

Additional utility settings.

 [Hierarchy Chart](#)

## ◆ **F u n c t i o n s**

---

- ◆ `inline void ErrorQuery( ViInt32& code, CString& msg )` Reads an error code and a message from the instrument error queue.
  - ◆ `inline void LockSession()` Gets a multithread lock on the instrument session.
  - ◆ `inline void Reset()` Resets the instrument to a known state.
  - ◆ `inline void SelfTest( ViInt32& code, CString& msg )` Runs the instrument self-test routine and returns the test result.
  - ◆ `inline void UnlockSession()` Releases a lock acquired on an instrument session using `LockSession`.
- 
-

# Measurement Studio IVI DMM Driver Overview

---

You use CNiIviDmm to control digital multimeters (DMMs) that have IVI class compliant instrument drivers.

## Top-Level Classes

---

[CNiIviDmm](#)



# SimpleIviDmm Example Program

This topic includes a summary of and links to the SimpleIviDmm example program. You can load the examples in either Microsoft Visual C++ 6.0 or Microsoft Visual C++ .NET. When you run an example, Measurement Studio Reference determines what version of Visual C++ you have installed and runs the example accordingly.

<b>Example</b>	<b>Description</b>	<b>Visual C++ 6.0</b>	<b>Visual C++ .NET</b>	<b>Run Example</b>
SimpleIviDmm	The SimpleIviDmm example demonstrates how to use the CNiIviDmm class and the SampleDmm virtual instrument driver to simulate measurements of DC volts.	<a href="#">Load</a>	<a href="#">Load</a>	<a href="#">Run</a>

# **CNiViDmm**



**Class**

Declared in:  
**NiViDmm.h**

## Overview

---

CNiIviDmm controls digital multimeters (DMMs) that have IVI class compliant instrument drivers. CNiIviDmm allows you to measure scalar quantities of an input signal with a DMM. Typically, the measured quantity is a voltage (AC or DC), current, or resistance. However, the CNiIviDmm class supports instruments that measure other quantities such as temperature and frequency. CNiIviDmm supports typical DMMs as well as common extended functionality found in more complex instruments.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

▲ CNiIviClassDriver

## ◆ Data Items

---

◆ <a href="#">CNIviDmmAc</a>	<b><u>Ac</u></b>	Additional settings for taking AC measurements.
◆ <a href="#">CNIviDmmAdvanced</a>	<b><u>Advanced</u></b>	Advanced Dmm properties.
◆ <a href="#">CNIviDmmFrequency</a>	<b><u>Frequency</u></b>	Additional settings for taking frequency and period measurements.
◆ <a href="#">long</a>	<b><u>Function</u></b>	The kind of measurement to take.
◆ <a href="#">CNIviDmmMeasurement</a>	<b><u>Measurement</u></b>	Functions for acquiring and retrieving measurements.
◆ <a href="#">double</a>	<b><u>Range</u></b>	The measurement range.
◆ <a href="#">double</a>	<b><u>Resolution</u></b>	The measurement resolution of the DMM in absolute units.
◆ <a href="#">CNIviDmmTemperature</a>	<b><u>Temperature</u></b>	Additional settings for taking temperature measurements.
◆ <a href="#">CNIviDmmTrigger</a>	<b><u>Trigger</u></b>	Additional settings for taking measurements after a trigger event occurs.

## ◆ Constructors

---

- ◆ `inline CNiIviDmm()` Default constructor.
- ◆ `inline CNiIviDmm( const CString &resourceName, bool idQuery = true, bool reset = true, const CString &options = "" )` Constructor.

## ◆ **F u n c t i o n s**

---

- ◆ `inline void Configure( long function, double range, double resolution )` Sets the Function, Range, and Resolution properties.
- 
-

# **CNiViDmmAc**



**Class**

Declared in:  
**NiViDmm.h**



## **Overview**

---

Additional settings for taking AC measurements. These settings affect the behavior of the instrument only if the Function property is set to an AC voltage or AC current measurement.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ double     FrequencyMax The maximum expected frequency component of the input signal in Hertz.
- ◆ double     FrequencyMin The minimum expected frequency component of the input signal in Hertz.

## ◆ **F u n c t i o n s**

---

- ◆ `inline void ConfigureBandwidth(` Configures the AC minimum and  
`double min, double` maximum frequency for DMMs that  
`max )` take AC voltage or AC current  
measurements.
-

# **CNiViDmmAdvanced**



**Class**

Declared in:  
**NiViDmm.h**

## Overview

---

Advanced Dmm properties.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ double	<u>ActualRange</u>	The actual range the Dmm is currently using, even when auto-ranging.
◆ double	<u>ApertureTime</u>	The measurement aperture time for the current configuration.
◆ long	<u>ApertureTimeUnit</u>	The units for the ApertureTime property.
◆ long	<u>AutoZero</u>	The auto-zero mode.
◆ double	<u>PowerlineFrequency</u>	The power line frequency in hertz.

---

# **CNiIviDmmFrequency**



**Class**

Declared in:  
**NiIviDmm.h**

## Overview

---

Additional settings for taking frequency and period measurements.

These settings affect the behavior of the instrument only if the Function property is set to a frequency or period measurement.

 [Hierarchy Chart](#)



## ◆ Data Items

---

- ◆ double     VoltageRange The expected maximum voltage value of the input signal.
- 
-

# **CNiIviDmmMeasurement**



**Class**

Declared in:  
**NiIviDmm.h**

## **Overview**

---

Functions for acquiring and retrieving measurements.

 [Hierarchy Chart](#)

## ◆ Functions

---

◆ inline void	<b><u>Abort()</u></b>	Aborts a previously initiated measurement.
◆ inline double	<b><u>Fetch( long int maxTime )</u></b>	Returns a previously acquired measured values.
◆ inline <u>CNiReal164Vector</u>	<b><u>FetchMultiPoint( long int maxTime )</u></b>	Returns a previously acquired vector containing the measured values.
◆ inline void	<b><u>Initiate()</u></b>	Initiates a measurement.
◆ inline bool	<b><u>IsOverRange( double measurementValue )</u></b>	Returns true when the input value is equal to Not-A-Number.
◆ inline double	<b><u>Read( long int maxTime )</u></b>	Initiates and acquires a measurement, and returns the measured value.
◆ inline <u>CNiReal164Vector</u>	<b><u>ReadMultiPoint( long int maxTime )</u></b>	Initiates and acquires a multi-point measurement, and returns a vector containing the measured values.
◆ inline void	<b><u>SendSoftwareTrigger()</u></b>	Sends a command to trigger the Dmm.

---

# **CNiViDmmRtd**



**Class**

Declared in:  
**NiViDmm.h**

## Overview

---

Additional settings for taking temperature measurements with a resistance temperature device. These settings affect the behavior of the instrument only if the `Function` property is set to `CNiIviDmm::TemperatureFunction` and the `Temperature.TransducerType` property is set to `CNiIviDmmTemperature::Rtd2WireTransducer` or `CNiIviDmmTemperature::Rtd4WireTransducer`.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ double Alpha The alpha parameter of the Rtd.
- ◆ double Resistance The R0 (reference value) parameter of the Rtd.

## ◆ **F u n c t i o n s**

---

- ◆ `inline void Configure( double alpha, double resistance )` Configures the alpha and resistance parameters for the Rtd.
- 
-



# **CNiIviDmmTemperature**



**Class**

Declared in:  
**NiIviDmm.h**

## Overview

---

Additional settings for taking temperature measurements. These settings affect the behavior of the instrument only if the `Function` property is set to `CNiIviDmm::TemperatureFunction`.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ long Count The number of trigger events to which to respond.
- ◆ long MeasurementCompleteDestination The destination of the signal the Dmm generates after each measurement.
- ◆ long SampleCount The number of measurements th Dmm takes each time it receives trigger.
- ◆ double SampleInterval The time in seconds between samples when the SampleTrigger property is set to CniIviDmm::IntervalSampleTrigger
- ◆ long SampleTrigger The sample trigger on which to take measurements when the SampleCount property is greater than 1.

## ◆ Functions

---

- ◆ `inline void Configure( long int count, long int sampleCount, long sampleTrigger, double sampleInterval )` Configures the Count, SampleCount, SampleTrigger, and SampleInterval properties.
- 
-

# **CNiIviDmmThermistor**



**Class**

Declared in:  
**NiIviDmm.h**

## Overview

---

Additional settings for taking temperature measurements with a thermistor. These settings affect the behavior of the instrument only if the `Function` property is set to `CNiIviDmm::TemperatureFunction` and the `Temperature.TransducerType` property is set to `CNiIviDmmTemperature::ThermistorTransducer`.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ double     Resistance     The resistance of the thermistor in Ohms.

---

---

# **CNiIviDmmThermocouple**



**Class**

Declared in:  
**NiIviDmm.h**



## Overview

---

Additional settings for taking temperature measurements with a thermocouple. These settings affect the behavior of the instrument only if the `Function` property is set to `CNiIviDmm::TemperatureFunction` and the `Temperature.TransducerType` property is set to `CNiIviDmm::ThermocoupleTransducer`.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ double      FixedRefJunction The external reference junction temperature in degrees Celsius when a fixed reference junction thermocouple is used to take the temperature measurement.
- ◆ long        RefJunctionType The type of reference junction to be used in the reference junction compensation.
- ◆ long        Type The type of thermocouple used to measure the temperature.

## ◆ **F u n c t i o n s**

---

- ◆ `inline void Configure( long type, long refJunctionType )` Configures the thermocouple type and the reference junction type.
- 
-

# **CNiIviDmmTrigger**



**Class**

Declared in:  
**NiIviDmm.h**

## Overview

---

Additional settings for taking measurements after a trigger event occurs.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ double                      **Delay**                      The amount of time in seconds to wait after a trigger occurs to actually take the measurement.
- ◆ CNiIviDmmMultiPoint **MultiPoint** Additional settings for taking multiple measurements on a trigger event.
- ◆ long                              **Slope**                              The direction of the edge of the trigger signal on which to take a measurement.
- ◆ long                              **Source**                              The source of the signal that causes the Dmm to take a measurement.

## ◆ Functions

---

- ◆ inline void **Configure**( long source, double delay ) Sets the Source and Delay properties.
-

# **Measurement Studio IVI Function Generator Driver Overview**

---

You use CNiIviFgen to control function generators that have IVI class compliant instrument drivers.

## **Top-Level Classes**

---

[CNiIviFgen](#)



# SimpleIviFgen Example Program

This topic includes a summary of and links to the SimpleIviFgen example program. You can load the examples in either Microsoft Visual C++ 6.0 or Microsoft Visual C++ .NET. When you run an example, Measurement Studio Reference determines what version of Visual C++ you have installed and runs the example accordingly.

<b>Example</b>	<b>Description</b>	<b>Visual C++ 6.0</b>	<b>Visual C++ .NET</b>	<b>Run Example</b>
SimpleIviFgen	The SimpleIviFgen example uses the CNiIviFgen class and the SampleFgen virtual instrument driver to simulate a function generator.	<a href="#">Load</a>	<a href="#">Load</a>	<a href="#">Run</a>

# **CNiIviFgen**



**Class**

Declared in:  
**NiIviFgen.h**

## Overview

---

CNiIviFgen controls function generators that have IVI class compliant instrument drivers. CNiIviFgen encapsulates the functionality of a function generator and can be applied to a wide range of instruments. The output signal is typically functional in nature. For example, the output signal is usually sinusoidal or square. Some instruments support the generation of arbitrary waveforms, which consist of user-specified data. If the function generator also supports the generation of arbitrary waveform sequences, the output signal can consist of a sequence of repeated arbitrary waveforms. CNiIviFgen supports typical function generators as well as common extended functionality found in more complex instruments.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

▲ CNiIviClassDriver

## ◆ Data Items

---

◆ <a href="#">CNIvifgenAm</a>	<b><u>Am</u></b>	Additional settings for generating Amplitude Modulated waveforms.
◆ <a href="#">CNIvifgenArbitrary</a>	<b><u>Arbitrary</u></b>	Additional settings for generating arbitrary waveforms and sequences.
◆ <a href="#">long</a>	<b><u>ChannelCount</u></b>	The number of channels that the specific driver supports.
◆ <a href="#">CNIvifgenFm</a>	<b><u>Fm</u></b>	Additional settings for generating Frequency Modulated waveforms.
◆ <a href="#">CNIvifgenOutput</a>	<b><u>Output</u></b>	Additional Settings related the generation of the output waveform.
◆ <a href="#">CNIvifgenStandardWaveform</a>	<b><u>StandardWaveform</u></b>	Additional settings for generating standard waveforms.
◆ <a href="#">CNIvifgenTrigger</a>	<b><u>Trigger</u></b>	Additional settings for configuring the trigger system.

## ◆ Constructors

---

- ◆ `inline CNilviEgen()` Default constructor.
- ◆ `inline CNilviEgen( const CString &resourceName, bool idQuery = true, bool reset = true, const CString &options = "" )` Constructor.



# **CNiIviFgenAm**



**Class**

Declared in:  
**NiIviFgen.h**



## Overview

---

Additional settings for generating Amplitude Modulated waveforms.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ bool	<b><u>Enabled[]</u></b>	Enables or disables amplitude modulation.
◆ double	<b><u>InternalDepth</u></b>	The extent of modulation as a percentage that the function generator applies to the carrier signal when the Source property is set to CNIiviFgen::AmInternalSource.
◆ double	<b><u>InternalFrequency</u></b>	The frequency in Hertz of the waveform that the function generator uses to modulate the output signal when the Source property is set to CNIiviFgen::AmInternalSource.
◆ long	<b><u>InternalWaveform</u></b>	The waveform that the function generator uses to modulate the output signal when the Source property is set to CNIiviFgen::AmInternalSource.
◆ long	<b><u>Source[]</u></b>	The signal that the function generator uses to modulate the output signal.

## ◆ Functions

---

- ◆ `inline void ConfigureInternal(  
double depth, long  
waveform, double  
frequency )` Configures the properties that  
control the function generator's  
internal amplitude modulation  
source.
- 
-

# **CNiIviFgenArbitrary**

**Class**



Declared in:  
**NiIviFgen.h**

## Overview

---

Additional settings for generating arbitrary waveforms and sequences. These settings affect the instrument behavior when the `Output.Mode` property is set to `CNiIviFgen::ArbitraryMode` or `CNiIviFgen::ArbitrarySequenceMode`.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ double	<u>Gain</u> []	The factor by which the function generator scales the arbitrary waveform data.
◆ double	<u>Offset</u> []	The value the function generator adds to the arbitrary waveform data.
◆ double	<u>SampleRate</u>	The rate in samples-per-second at which to generate the points in arbitrary waveforms.
◆ <u>CNiIviFgenArbitrarySequence</u>	<u>Sequence</u>	Additional settings for generating arbitrary sequences.
◆ <u>CNiIviFgenArbitraryWaveform</u>	<u>Waveform</u>	Additional settings for generating arbitrary waveforms.

## ◆ **F u n c t i o n s**

---

- ◆ `inline void ClearMemory()` Removes all previously created arbitrary waveforms and sequences from the function generator's memory.
-

# **CNiIviFgenArbitrarySequence**

**Class**

Declared in:  
**NiIviFgen.h**



## Overview

---

Additional settings for generating arbitrary sequences. These settings affect the instrument behavior when the `Output.Mode` property is set to `CNiIviFgen::ArbitrarySequenceMode`.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ long	<u>Handle[]</u>	
◆ long int	<u>LengthMax</u>	The maximum number of arbitrary waveforms the function generator allows in a sequence.
◆ long int	<u>LengthMin</u>	The minimum number of arbitrary waveforms the function generator allows in a sequence.
◆ long int	<u>LoopCountMax</u>	The maximum number of times the function generator can repeat a waveform in a sequence.
◆ long int	<u>NumberSequencesMax</u>	The maximum number of arbitrary sequences the function generator allows.

## ◆ Functions

---

- ◆ `inline void` **Clear**( `long` handle ) Removes a previously created arbitrary sequence from the function generator's memory.
  - ◆ `inline void` **Configure**( `const CString&` ch, `long` handle, `double` gain, `double` offset ) Configures the properties of the function generator that affect arbitrary sequence generation.
  - ◆ `inline ViInt32` **Create**( `const CNiInt32Vector&` handles, `const CNiInt32Vector&` loops ) Creates an arbitrary sequence from an array of waveform handles and an array of corresponding loop counts, and return a handle that identifies the sequence.
-

# **CNiIviFgenArbitraryWaveform**

**Class**

Declared in:  
**NiIviFgen.h**

## Overview

---

Additional settings for generating arbitrary waveforms. These settings affect the instrument behavior when the `Output.Mode` property is set to `CNiIviFgen::ArbitraryMode`.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ double	<u>Frequency[]</u>	The rate in waveforms-per-second at which to generate an entire arbitrary waveform.
◆ long	<u>Handle[]</u>	
◆ long int	<u>NumberWaveformsMax</u>	The maximum number of arbitrary waveforms that the function generator allows.
◆ long int	<u>Quantum</u>	The value of which the length of all arbitrary waveforms must be a multiple.
◆ long int	<u>SizeMax</u>	The maximum number of points the function generator allows in an arbitrary waveform.
◆ long int	<u>SizeMin</u>	The minimum number of points the function generator allows in an arbitrary waveform.

## ◆ Functions

---

- ◆ inline void **Clear**( long handle ) Removes a previously created arbitrary waveform from the function generator's memory.
  - ◆ inline void **Configure**( const CString& ch, long handle, double gain, double offset ) Configures the properties of the function generator that affect arbitrary waveform generation.
  - ◆ inline ViInt32 **Create**( const CNiReal64Vector& wfm ) Creates an arbitrary waveform and returns a handle that identifies that waveform.
-

# **CNiIviFgenFm**



**Class**

Declared in:  
**NiIviFgen.h**



## Overview

---

Additional settings for generating Frequency Modulated waveforms.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ bool	<u>Enabled</u> []	Specifies whether the function generator applies frequency modulation to the output signal.
◆ double	<u>InternalDeviation</u>	The maximum frequency deviation in Hertz that the modulating waveform applies to the carrier waveform when the Source property is set to CNIiviFgen::FmInternalSource.
◆ double	<u>InternalFrequency</u>	The frequency in Hertz of the waveform that the function generator uses to modulate the output signal when the Source property is set to CNIiviFgen::FmInternalSource.
◆ long	<u>InternalWaveform</u>	The waveform that the function generator uses to modulate the output signal when the Source property is set to CNIiviFgen::FmInternalSource.
◆ long	<u>Source</u> []	The signal that the function generator uses to modulate the output signal.

## ◆ Functions

---

- ◆ `inline void ConfigureInternal(  
double deviation, long  
waveform, double  
frequency )` Configures the properties that  
control the function generator's  
internal frequency modulation  
source.
- 
-

# **CNiIviFgenOutput**



**Class**

Declared in:  
**NiIviFgen.h**

## Overview

---

Additional Settings related the generation of the output waveform.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ bool	<u>Enabled</u> []	Specifies whether the generated signal appears at the output connector.
◆ double	<u>Impedance</u> []	The function generator's output impedance at the output connector.
◆ long	<u>Mode</u>	Determines the kind of waveform the function generator produces.
◆ long	<u>OperationMode</u> []	How the function generator produces waveforms.
◆ long	<u>RefClockSource</u>	The reference clock source from which the function generator derives frequencies and sample rates for generating waveforms.

---

# **CNiIviFgenStandardWaveform**

**Class**

Declared in:  
**NiIviFgen.h**

## Overview

---

Additional settings for generating standard waveforms. These settings affect the instrument behavior when the `Output.Mode` property is set to `CNiIviFgen::FunctionMode`.

 [Hierarchy Chart](#)



## ◆ Data Items

---

- ◆ double     Amplitude[]     The waveform's peak-to-peak amplitude in Volts.
- ◆ double     DcOffset[]     The waveform's DC Voltage offset from ground to center.
- ◆ double     DutyCycleHigh[]     The percentage of one waveform cycle that the output voltage level remains high when the waveform property is set to `CNiIviFgen::SquareWaveform`.
- ◆ double     Frequency[]     The waveform's frequency in Hertz.
- ◆ double     StartPhase[]     The waveform's horizontal offset in degrees.
- ◆ long        Waveform[]     The standard waveform to generate.

## ◆ Functions

---

- ◆ `inline void Configure( const CString& ch, long wfm, double amplitude, double dcOffset, double frequency, double startPhase )` Configures the properties of the function generator that affect standard waveform generation.
- 
-

# **CNiIviFgenTrigger**



**Class**

Declared in:  
**NiIviFgen.h**

## Overview

---

Additional settings for configuring the trigger system.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ long int     **BurstCount**[] The number of waveform cycles to generate after receiving a trigger when the `Output.OperationMode` property is set to `CNiIviFgen::BurstOperationMode`.
- ◆ double     **InternalRate** The rate in triggers-per-second at which the internal trigger source produces a trigger.
- ◆ long     **Source**[] The trigger source upon which to generate a signal, based on the operation mode.

## ◆ **F u n c t i o n s**

---

- ◆ `inline void SendSoftwareTrigger()` Sends a command to trigger the function generator.
- 
-

# **Measurement Studio IVI Oscilloscope Driver Overview**

---

You use CNiIviScope to control oscilloscopes that have IVI class compliant instrument drivers.

## **Top-Level Classes**

---

[CNiIviScope](#)

# SimpleIviScope Example Program

This topic includes a summary of and links to the SimpleIviScope example program. You can load the examples in either Microsoft Visual C++ 6.0 or Microsoft Visual C++ .NET. When you run an example, Measurement Studio Reference determines what version of Visual C++ you have installed and runs the example accordingly.

<b>Example</b>	<b>Description</b>	<b>Visual C++ 6.0</b>	<b>Visual C++ .NET</b>	<b>Run Example</b>
SimpleIviScope	The SimpleIviScope example uses the CNiIviScope class and the SampleScope virtual instrument driver to simulate an oscilloscope read.	<a href="#">Load</a>	<a href="#">Load</a>	<a href="#">Run</a>



# **CNiIviScope**



**Class**

Declared in:  
**NiIviScope.h**

## Overview

---

CNiIviScope controls oscilloscopes that have IVI class compliant instrument drivers. CNiIviScope allows you to acquire a voltage waveform from an analog input signal with an oscilloscope. CNiIviScope acquires the points in the waveform at a configurable interval and can acquire the points sequentially in real-time sampling or interleaved from multiple waveform acquisitions in equivalent time sampling. CNiIviScope allows you to use a property of one of the input signals, typically a rising or falling edge, to trigger the acquisition. In addition, CNiIviScope supports instruments that have more complex acquisition modes such as average, envelope, and peak detect. CNiIviScope also supports trigger types such as TV, runt, and glitch. CNiIviScope supports the typical oscilloscope as well as common extended functionality found in more complex instruments.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

▲ CNiIviClassDriver

## ◆ Data Items

---

◆ <a href="#">CNIviScopeAcquisition</a>	<b><u>Acquisition</u></b>	Additional settings for configuring the way the oscilloscope performs an acquisition.
◆ <a href="#">CNIviScopeChannel</a>	<b><u>Channel</u></b>	Additional settings for configuring the oscilloscope's acquisition channels.
◆ long	<b><u>ChannelCount</u></b>	
◆ <a href="#">CNIviScopeMeasurement</a>	<b><u>Measurement</u></b>	Functions for acquiring waveforms and waveform measurements.
◆ <a href="#">CNIviScopeReferenceLevel</a>	<b><u>ReferenceLevel</u></b>	Additional settings for configuring the references levels the oscilloscope uses for waveform measurements.
◆ <a href="#">CNIviScopeTrigger</a>	<b><u>Trigger</u></b>	Additional settings for configuring the trigger.

## ◆ Constructors

---

- ◆ `inline CNilviScope()` Default constructor.
- ◆ `inline CNilviScope( const CString &resourceName, bool idQuery = true, bool reset = true, const CString &options = "" )` Constructor.

## ◆ **Functions**

---

◆ `inline` `CString` `GetChannelName`( ViInt32 index ) `const`

---

---

# **CNiViScopeAcquisition**



**Class**

Declared in:  
**NiViScope.h**

## Overview

---

Additional settings for configuring the way the oscilloscope performs an acquisition.

 [Hierarchy Chart](#)



## ◆ Data Items

---

◆ long	<u>Averages</u>	Specifies the number of waveforms the oscilloscope acquires and averages when the Type property is set to <code>CNiIviScope::Average</code> .
◆ long	<u>Envelopes</u>	Specifies the number of waveforms the oscilloscope acquires when the Type property is set to <code>CNiIviScope::Envelope</code> .
◆ long	<u>Interpolation</u>	Specifies the interpolation method the oscilloscope uses when it cannot sample a voltage for every point in the waveform record.
◆ long	<u>MinNumPts</u>	Specifies the minimum number of points you require in the waveform record for each channel.
◆ long	<u>RecordLength</u>	Returns the actual number of points the oscilloscope acquires for each channel.
◆ long	<u>SampleMode</u>	Returns a value indicating the mode in which the instruments takes samples.
◆ double	<u>SampleRate</u>	Returns the effective digitizing rate in sample per second.
◆ double	<u>StartTime</u>	Specifies the length of time in seconds from the trigger event to the first point in the waveform record.
◆ double	<u>TimePerRecord</u>	Specifies the time in seconds that corresponds to the record length.
◆ long	<u>Type</u>	Specifies how the oscilloscope acquires data and fills the waveform record.

## ◆ **F u n c t i o n s**

---

- ◆ `inline void ConfigureRecord(  
ViReal64 timePerRecord,  
ViInt32 minNumPts,  
ViReal64 startTime )` Configures the most commonly  
configured properties of the  
oscilloscope acquisition  
subsystem.
- 
-

# **CNiViScopeChannel**



**Class**

Declared in:  
**NiViScope.h**

## Overview

---

Additional settings for configuring the oscilloscope's acquisition channels.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ long	<u>Coupling</u> []	Specifies how the oscilloscope couples the input signal for the channel.
◆ bool	<u>Enabled</u> []	Specifies whether the oscilloscope acquires a waveform or measurement for a channel.
◆ double	<u>InputFrequencyMax</u> []	Specifies the maximum input frequency in hertz of the channel.
◆ double	<u>InputImpedance</u> []	Specifies the input impedance in ohms for the channel.
◆ double	<u>Offset</u> []	Specifies the location in volts relative to ground of the center of the range that you specify with the Range property.
◆ double	<u>ProbeAttenuation</u> []	Specifies the scaling factor by which the probe you attach to the channel attenuates the input.
◆ double	<u>ProbeSense</u> []	Returns the probe attenuation value the oscilloscope automatically senses.
◆ double	<u>Range</u> []	Specifies the absolute value in volts of the input range the oscilloscope can acquire for the channel.

## ◆ Functions

---

- ◆ `inline void Configure( const CString &channel, double range, double offset, long coupling, double probeAttenuation, bool enabled )` Configures the most commonly configured properties of the oscilloscope channel subsystem.
  - ◆ `inline void ConfigureCharacteristics( const CString &channel, double inputImpedance, double maxInputFrequency )` Configures the properties that control the electrical characteristics of the channel.
-

# **CNiViScopeMeasurement**



**Class**

Declared in:  
**NiViScope.h**

## **Overview**

---

Functions for acquiring waveforms and waveform measurements.

 [Hierarchy Chart](#)



## ◆ Functions

---

- ◆ inline void **Abort()** Aborts an acquisition and returns the oscilloscope to the Idle state.
- ◆ inline void **AutoSetup()** Automatically configures the instrument.
- ◆ inline void **FetchWaveform**( const CString &channel, CNiReal64Vector &waveform, double &xFirst, double &xIncrement ) This function returns the waveform the oscilloscope acquires for the channel you specify.
- ◆ inline double **FetchWaveformMeasurement**( Fetches a waveform measurement from the channel you specify The waveform on which the oscilloscope calculates the waveform measurement is from an acquisition that you previously initiated.  
const CString &channel, long measurementFunction )
- ◆ inline void **FetchWaveformMinMax**( This function returns the waveforms the oscilloscope acquires for the channel you specify.  
const CString &channel, CNiReal64Vector &minWaveform, CNiReal64Vector &maxWaveform, double &xFirst, double &xIncrement )
- ◆ inline void **Initiate()** Initiates a waveform acquisition.
- ◆ inline bool **IsWaveformElementInvalid**( This function determines whether a value you pass from the waveform array is invalid.  
double elementValue )
- ◆ inline void **ReadWaveform**( Initiates an acquisition on all channels that you enable with the Channel.Configure  
const CString &channel, long int maxTime, CNiReal64Vector &waveform, double &xFirst, double

&xIncrement ) function.

- ◆ inline double **ReadWaveformMeasurement**(  
const CString &channel, long  
int maxTime, long  
measurementFunction ) Initiates an acquisition on  
all channels that you  
enable with the  
Channel.Configure  
function.
  - ◆ inline void **ReadWaveformMinMax**(  
const CString &channel, long  
int maxTime, CNiReal64Vector  
&minWaveform,  
CNiReal64Vector  
&maxWaveform, double  
&xFirst, double &xIncrement ) Initiates an acquisition on  
all channels that you  
enable with the  
Channel.Configure  
function.
  - ◆ inline long **Status**() Returns a value indicating  
the state of an acquisition.
-

# **CNiIviScopeReferenceLevel**

**Class**

Declared in:  
**NiIviScope.h**

## **Overview**

---

Additional settings for configuring the references levels the oscilloscope uses for waveform measurements.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ **double**     **High** Specifies the high reference the oscilloscope uses for waveform measurements.
- ◆ **double**     **Low** Specifies the low reference the oscilloscope uses for waveform measurements.
- ◆ **double**     **Mid** Specifies the mid reference the oscilloscope uses for waveform measurements.

## ◆ **F u n c t i o n s**

---

- ◆ `inline void Configure( double low, double mid, double hi )` Configures the reference levels for waveform measurements.
- 
-

# **CNiViScopeTrigger**



**Class**

Declared in:  
**NiViScope.h**

## Overview

---

Additional settings for configuring the trigger.

 [Hierarchy Chart](#)



## ◆ Data Items

---

◆ <a href="#">CNiIviScopeTriggerAcLine</a>	<b><u>AcLine</u></b>	Additional settings for configuring the ACLine trigger.
◆ <a href="#">bool</a>	<b><u>Continuous</u></b>	Specifies whether the oscilloscope continuously initiates waveform acquisition.
◆ <a href="#">long</a>	<b><u>Coupling</u></b>	Specifies how the oscilloscope couples the trigger source.
◆ <a href="#">CNiIviScopeTriggerEdge</a>	<b><u>Edge</u></b>	Additional settings for configuring the edge trigger.
◆ <a href="#">CNiIviScopeTriggerGlitch</a>	<b><u>Glitch</u></b>	Additional settings for configuring the glitch trigger.
◆ <a href="#">double</a>	<b><u>Holdoff</u></b>	Specifies the length of time in seconds that the oscilloscope waits after it detects a trigger until it enables the trigger subsystem to detect another trigger.
◆ <a href="#">double</a>	<b><u>Level</u></b>	Specifies the voltage threshold in volts for the trigger subsystems.
◆ <a href="#">long</a>	<b><u>Modifier</u></b>	Specifies the trigger modifier.
◆ <a href="#">CNiIviScopeTriggerRunt</a>	<b><u>Runt</u></b>	Additional settings for configuring the runt trigger.
◆ <a href="#">CString</a>	<b><u>Source</u></b>	Specifies the source the oscilloscope monitors for a trigger.
◆ <a href="#">CNiIviScopeTriggerTv</a>	<b><u>Tv</u></b>	Additional settings for configuring the TV trigger.
◆ <a href="#">long</a>	<b><u>Type</u></b>	Specifies the trigger type.
◆ <a href="#">CNiIviScopeTriggerWidth</a>	<b><u>Width</u></b>	Additional settings for configuring the width trigger.

## ◆ **F u n c t i o n s**

---

- ◆ `inline void Configure( long type, double holdoff )` Configures the trigger type and holdoff.
- 
-

# **CNiIviScopeTriggerAcLine**



**Class**

Declared in:  
**NiIviScope.h**

## Overview

---

Additional settings for configuring the ACLine trigger. These settings affect instrument operation only when the `Trigger.Type` property is set to `CNiIviScope::AcLineTrigger`.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ long      **Slope** Specifies the slope of the zero crossing upon which the scope triggers.
- 
-

# **CNiViScopeTriggerEdge**



**Class**

Declared in:  
**NiViScope.h**

## Overview

---

Additional settings for configuring the edge trigger. These settings affect instrument operation only when the `Trigger.Type` property is set to `CNiIviScope::EdgeTrigger`.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ long      **Slope** Specifies whether a rising or a falling edge triggers the oscilloscope.



## ◆ Functions

---

- ◆ inline void **Configure**( const **CString** &source, **double** level, **long** slope )      Configures the edge trigger.
- 
-

# **CNiViScopeTriggerGlitch**

**Class**

Declared in:  
**NiViScope.h**

## Overview

---

Additional settings for configuring the glitch trigger. These settings affect instrument operation only when the `Trigger.Type` property is set to `CNiIviScope::GlitchTrigger`.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ long      **Condition** Specifies the glitch condition that triggers the oscilloscope.
- ◆ long      **Polarity** Specifies the polarity of the glitch that triggers the oscilloscope.
- ◆ double    **Width** Specifies the glitch width in seconds.

## ◆ Functions

---

- ◆ inline void **Configure**( const CString &source, double level, double width, long polarity, long condition ) Configures the glitch trigger.
- 
-

# **CNiIviScopeTriggerRunt**



**Class**

Declared in:  
**NiIviScope.h**

## Overview

---

Additional settings for configuring the runt trigger. These settings affect instrument operation only when the `Trigger.Type` property is set to `CNiIviScope::RuntTrigger`.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ long      **Polarity**      Specifies the polarity of the runt that triggers the oscilloscope.
- ◆ double    **ThresholdHigh** Specifies the high threshold in volts.
- ◆ double    **ThresholdLow**  Specifies the low threshold in volts.



## ◆ Functions

---

- ◆ inline void **Configure**( const `CString` &source, double lowThreshold, double highThreshold, long polarity )      Configures the runt trigger.
- 
-

# **CNiViScopeTriggerTv**



**Class**

Declared in:  
**NiViScope.h**

## Overview

---

Additional settings for configuring the TV trigger. These settings affect instrument operation only when the `Trigger.Type` property is set to `CNiIviScope::TvTrigger`.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ long      **Event**      Specifies the event on which the oscilloscope triggers.
- ◆ long      **LineNumber**      Specifies the line on which the oscilloscope triggers when the Event property is set to CNIviScope::TvLine.
- ◆ long      **Polarity**      Specifies the polarity of the TV signal.
- ◆ long      **SignalFormat**      Specifies the format of the TV signal on which the oscilloscope triggers.

## ◆ Functions

---

- ◆ inline void **Configure**( const `CString` &source, long signalFormat, long event, long polarity ) Configures the oscilloscope for TV triggering.
- 
-

# **CNiIviScopeTriggerWidth**



**Class**

Declared in:  
**NiIviScope.h**

## Overview

---

Additional settings for configuring the width trigger. These settings affect instrument operation only when the `Trigger.Type` property is set to `CNiIviScope::WidthTrigger`.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ long      **Condition**      Specifies whether a pulse that is inside or outside the high and low thresholds triggers the oscilloscope.
- ◆ long      **Polarity**      Specifies the polarity of the pulse that triggers the oscilloscope.
- ◆ double    **ThresholdHigh**    Specifies the high width threshold time in seconds.
- ◆ double    **ThresholdLow**    Specifies the low width threshold time in seconds.



## ◆ Functions

---

- ◆ inline void **Configure**( const **CString** &source, **double** level, **double** lowThreshold, **double** highThreshold, **long** polarity, **long** condition ) Configures the width trigger.
- 
-

# Measurement Studio IVI Switch Driver Overview

---

You use CNiIviSwth to control switches that have IVI class compliant instrument drivers.

## Top-Level Classes

---

[CNiIviSwth](#)

# SimpleIviSwitch Example Program

This topic includes a summary of and links to the SimpleIviSwitch example program. You can load the examples in either Microsoft Visual C++ 6.0 or Microsoft Visual C++ .NET. When you run an example, Measurement Studio Reference determines what version of Visual C++ you have installed and runs the example accordingly.

<b>Example</b>	<b>Description</b>	<b>Visual C++ 6.0</b>	<b>Visual C++ .NET</b>	<b>Run Example</b>
SimpleIviSwitch	The SimpleIviSwitch example uses the CNiIviSwch class and the SampleSwitch virtual instrument driver to simulate connecting and disconnecting multiple switch paths using an IVI class driver.	<a href="#">Load</a>	<a href="#">Load</a>	<a href="#">Run</a>

# **CNiIviSwtch**



**Class**

Declared in:  
**NilviSwtch.h**

## Overview

---

CNiIviSwTch controls switches that have IVI class compliant instrument drivers. CNiIviSwTch encapsulates the functionality of a switch. Use CNiIviSwTch to establish a connection between two I/O channels. CNiIviSwTch supports instruments that perform triggered scanning. CNiIviSwTch supports typical switches as well as common extended functionality found in more complex switch instruments.

 [Hierarchy Chart](#)

## ▲ Base Classes

---

▲ CNiIviClassDriver

## ◆ Data Items

---

- ◆ [CNIvISwtchChannel](#) **Channel** Additional settings that affect the switch's channels.
- ◆ [long](#) **ChannelCount** The number of channels the specific instrument supports.
- ◆ [CNIvISwtchPath](#) **Path** Additional settings related to channel paths.
- ◆ [CNIvISwtchScan](#) **Scan** Additional settings for configuring a scanning switch.

## ◆ Constructors

---

- ◆ `inline CNiIviSwrch()` Default constructor.
- ◆ `inline CNiIviSwrch( const CString &resourceName, bool idQuery = true, bool reset = true, const CString &options = "" )` Constructor.



## ◆ **Functions**

---

◆ `inline` `CString` `GetChannelName`( `ViInt32` index ) `const`

---

---

# **CNiIviSwtchChannel**



**Class**

Declared in:  
**NilviSwtch.h**

## Overview

---

Additional settings that affect the switch's channels.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ <u>CNiIviSwtchCharacteristics</u>	<u>Characteristics</u>	Additional settings that affect the characteristic of the switch module.
◆ bool	<u>IsConfigurationChannel</u> []	Specifies whether to reserve the Channel for internal path creation.
◆ bool	<u>IsSourceChannel</u> []	Specifies whether you want to identify the channel as a source channel.

---

# **CNiIviSwtchCharacteristics**

**Class**

Declared in:  
**NiIviSwtch.h**

## Overview

---

Additional settings that affect the characteristic of the switch module.

 [Hierarchy Chart](#)

## ◆ Data Items

---

- ◆ double [AcCurrentCarryMax\[\]](#) The maximum AC current the channel can carry in Amperes RMS.
- ◆ double [AcCurrentSwitchingMax\[\]](#) The maximum AC current the channel can switch in Amperes RMS.
- ◆ double [AcPowerCarryMax\[\]](#) The maximum AC power the channel can carry in Volt-Amperes.
- ◆ double [AcPowerSwitchingMax\[\]](#) The maximum AC power the channel can switch in Volt-Amperes.
- ◆ double [AcVoltageMax\[\]](#) The maximum AC voltage the channel can switch in Volts RMS.
- ◆ double [Bandwidth\[\]](#) The channel's bandwidth in Hertz.
- ◆ double [DcCurrentCarryMax\[\]](#) The maximum DC current the channel can carry in Amperes.
- ◆ double [DcCurrentSwitchingMax\[\]](#) The maximum DC current the channel can switch in Amperes.
- ◆ double [DcPowerCarryMax\[\]](#) The maximum DC power the channel can carry in Watts.
- ◆ double [DcPowerSwitchingMax\[\]](#) The maximum DC power the channel can switch in Watts.
- ◆ double [DcVoltageMax\[\]](#) The maximum DC voltage the channel can switch in Volts.
- ◆ double [Impedance\[\]](#) The channel's impedance in Ohms.
- ◆ double [SettlingTime\[\]](#) The maximum length of time in seconds from after you make a connection until the signal flowing through the channel settles.

◆

long

**WireMode[]**

The switch module's wire mode.

---



# **CNiIviSwtchPath**



**Class**

Declared in:  
**NilviSwtch.h**

## Overview

---

Additional settings related to channel paths.

 [Hierarchy Chart](#)

## ◆ Functions

---

- ◆ inline long     **CanConnect**( const CString &ch1, const CString &ch2 ) const     Returns a value indicating whether a path is currently available between two channels given the existing connections.
  
  - ◆ inline void     **Connect**( const CString &ch1, const CString &ch2 )     Connects the shortest possible path between two channels.
  
  - ◆ inline void     **Disconnect**( const CString &ch1, const CString &ch2 )     Destroys a previously connected path between two channels.
  
  - ◆ inline void     **DisconnectAll**()     Disconnects all existing paths.
  
  - ◆ inline CString **GetPath**( const CString &ch1, const CString &ch2 ) const     Returns a string that uniquely identifies a path created with the Connect function.
  
  - ◆ inline bool     **IsDebounced**() const     Returns a whether all the previously created paths have settled.
  
  - ◆ inline void     **SetPath**( const CString &path )     Connects two channels by establishing the exact path specified with the path parameter.
  
  - ◆ inline void     **WaitForDebounce**( long maxTime ) const     Returns process control back to the user only after all the previously created paths have settled.
-

# **CNiIviSwtchScan**



**Class**

Declared in:  
**NilviSwtch.h**

## Overview

---

Additional settings for configuring a scanning switch.

 [Hierarchy Chart](#)

## ◆ Data Items

---

◆ long	<u>AdvancedOutput</u>	The method to use to notify another instrument that all signals going through the switch module have settled following the processing of one entry in the scan list.
◆ bool	<u>Continuous</u>	Specifies whether the switch module continues scanning from the top of the scan list after reaching the end of the list.
◆ double	<u>Delay</u>	The minimum amount of time in seconds that the switch module waits before it asserts the scan advanced output trigger after opening or closing the switch.
◆ long	<u>Input</u>	The source of the trigger for which the switch module waits when processing a scan list.
◆ <u>CString</u>	<u>List</u>	The scan list is a string that specifies channel connections and trigger conditions.
◆ long	<u>Mode</u>	Specifies what happens to existing connections that conflict with the connections you make in a scan list.
◆ long	<u>NumberOfColumns</u>	Returns the number of columns of a matrix or scanner.
◆ long	<u>NumberOfRows</u>	Returns the number of rows of a matrix or scanner.

## ◆ Functions

---

- ◆ inline void **Abort()** Aborts a previously initiated scan.
  - ◆ inline void **ConfigureList( const CString &List, long mode )** Configures the switch module for scanning.
  - ◆ inline void **ConfigureTrigger( double scanDelay, long triggerInput, long scanAdvancedOutput )** Configures the scan triggers for the scan list you establish with `ConfigureList` function.
  - ◆ inline void **Initiate()** Initiates a scan.
  - ◆ inline bool **IsScanning() const** Returns whether the switch module is scanning.
  - ◆ inline void **SendSoftwareTrigger()** Sends a command to trigger the switch.
  - ◆ inline void **WaitForScanComplete( long maxTime ) const** Waits until the instrument stops scanning.
-