**User Interface Common Component Overview**

The User Interface Common component contains class declarations that are shared by the User Interface component and the 3DGraph component.

**Note**

Because the User Interface component and 3DGraph component include ActiveX controls that link to the MFC DLL, projects that you design to use Measurement Studio User Interface or 3DGraph controls cannot link to static MFC.

**Top-Level Classes**

**CNiValuePair** - CNiValuePair objects configure an individual value pair, which consists of a name and a value. You use value pairs on the axes of knob, slide, or graph controls to associate a symbolic name with a value on the axis.
## User Interface Common Example Programs

This topic includes summaries of and links to the example programs associated with the User Interface Common component.

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
<th>Load example in VC++</th>
<th>Run example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bound Graph</strong></td>
<td>The Bound Graph example demonstrates binding a Graph to a DataSocket source and automatically graphing data without writing a single line of code.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic Bound Graph</strong></td>
<td>The Dynamic Bound Graph example demonstrates dynamically creating a property binding on a graph object.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Control Metrics</strong></td>
<td>The Control Metrics example demonstrates how to use the Control Metrics property of knob and slide controls.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic Move or Resize UI</strong></td>
<td>The Dynamic Move or Resize UI sample demonstrates how to dynamically change the size and position of a control.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Printing</strong></td>
<td>The Printing example demonstrates how to size and print an image of a graph control.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Simple UI</strong></td>
<td>The Simple UI example demonstrates the basic features of a Measurement Studio user interface.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>UI Formats</strong></td>
<td>The UI Formats example demonstrates various UI formats.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Class

Declared in:

NiControl.h
Overview

CNiControl is an intermediate class from which all Measurement Studio ActiveX control objects, such as CNiGraph, CNiSlide, and CNiKnob, derive.

Hierarchy Chart
Base Classes

CWnd
Constructors

CNiControl(CNiInterface::ThreadAccess threadAccess)  Default constructor.
Destructors

- ~CNiControl() Destructor.
Functions

- virtual void * GetCustomInterface(bool *pDidAddRef) Returns the custom interface pointer associated with the object.

- CNiInterface::ThreadAccess GetThreadAccess() Returns the thread access option to which the control is configured.

- virtual void InitCustomInterface() Initializes the custom interface pointer to associate with the object.

- virtual void PreSubclassWindow() Called by the MFC framework before the window is created.

- virtual void ReleaseCustomInterface() Releases the custom interface pointer associated with the object.

- virtual void ValidateControl() Validates the current state of the control.

- static long __stdcall WndProc(HWND hWnd,
UINT message, WPARAM wParam, LPARAM lParam )

MFC framework to process windows messages.
Class

Declared in:
NiValuePair.h
Overview

The `CNiValuePair` object configures an individual value pair, which consists of a name and a value. Use value pairs on the axes of knob, slide, or graph controls to associate a symbolic name with a value on the axis.

You also can use value pairs on the slide and knob control to implement a value-pairs-only control, which limits the valid values of the control to these value pairs.

You must initialize a `CNiValuePair` object from an existing object. If you do not initialize a `CNiValuePair` object from an existing object, a `CNiObjectNotInUsableState` exception will be thrown when you attempt to manipulate the instance of the `CNiValuePair`.

Note: To specify a date/time value, you must convert your date or time value to a double. A date is implemented as a floating-point value with the integer part of the number measuring days from midnight, 30 December 1899, and the fractional part representing the time of day. The absolute value of the fractional part of the number represents the time as a fraction of a day. Thus, 1 second equals 1 / 24 hours / 60 minutes, which is 1/86400 or approximately 1.157407e-5. So, midnight, 31 December 1899, is represented by 1.0. Similarly, 6 AM, 1 January 1900, is represented by 2.25, and midnight, 29 December 1899, is -1.0. However, 6 AM, 29 December 1899, is -1.25.

Hierarchy Chart
Base Classes

CNiInterface
<table>
<thead>
<tr>
<th><strong>Data Items</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CString</strong></td>
</tr>
<tr>
<td><strong>double</strong></td>
</tr>
</tbody>
</table>
Constructors

- **CNiValuePair()**
  Default constructor.

- **CNiValuePair(CWValuePair_CI* pCustom, CNiInterface::ThreadAccess option)**
  Constructor that attaches to the specified CWValuePair_CI pointer.

- **CNiValuePair(const CNiValuePair& source)**
  Copy constructor.
Destructors

~CNiValuePair

Destructor.
### Functions

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static const IID &amp; GetIid()</code></td>
<td>Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.</td>
</tr>
<tr>
<td><code>const CNiValuePair &amp; operator =( const CNiValuePair&amp; source )</code></td>
<td>Assignment operator.</td>
</tr>
</tbody>
</table>
Class

Declared in:
NiValuepairs.h
Overview

A CNiValuePairs object is a collection of value pairs on a CNiAxis or CNiAxis3D object.

- Use the CNiAxis::ValuePairs or CNiAxis3D::ValuePairs property to obtain the associated value pair collection.
- Use the Add function to create additional value pairs. Add returns a CNiValuePair object, which represents the new value pair.
- Use the Item function to access existing value pairs in the collection. This function can access value pairs by either name or index.
- Use the Remove function to remove existing value pairs from the collection. This function can access value pairs by either name or index.
- Use the RemoveAll function to remove all value pairs from the collection.

Use the properties listed below to control the appearance of value pairs.

- GridLines
- LabelType
- Location
- MajorTicks

Hierarchy Chart
Base Classes

CNiInterface
### Data Items

<table>
<thead>
<tr>
<th>Type</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>Count</td>
<td>Returns the number of value pairs in the collection.</td>
</tr>
<tr>
<td>bool</td>
<td>GridLines</td>
<td>Specifies if grid lines are drawn at value pair locations.</td>
</tr>
<tr>
<td>ValuePairLabels</td>
<td>LabelType</td>
<td>Specifies the type of labels to draw for the value pairs.</td>
</tr>
<tr>
<td>ValuePairLocations</td>
<td>Location</td>
<td>Specifies if value pairs are placed on the axis by their value or by their index.</td>
</tr>
<tr>
<td>bool</td>
<td>MajorTicks</td>
<td>Specifies if major ticks are placed at the location of the value pairs.</td>
</tr>
</tbody>
</table>
Constructors

- **CNiValuePairs()**
  Default constructor.

- **CNiValuePairs(CWValuePairs_CI* pCustom, CNiInterface::ThreadAccess option )**
  Constructor that attaches to the specified CWValuePairs_CI pointer.

- **CNiValuePairs(const CNiValuePairs& source )**
  Copy constructor.
Destructors

~CNiValuePairs() Destructor.
 Functions

- **CNiValuePair**
  - **Add()**
    - Adds a value pair to the collection and returns the new value pair.

- **static const IID &**
  - **GetIid()**
    - Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.

- **CNiValuePair**
  - **Item(const CString & valuePairName)**
    - Returns the specified value pair from the collection.
  - **Item(long valuePairIndex)**
    - Returns the specified value pair from the collection.

- **const CNiValuePairs &**
  - **operator = (const CNiValuePairs & source)**
    - Assignment operator.

- **void**
  - **Remove(const CString & valuePairName)**
    - Removes the specified value pair from the collection.
  - **Remove(long valuePairIndex)**
    - Removes the specified value pair from the collection.

- **void**
  - **RemoveAll()**
    - Removes all value pairs from the collection.

- **void**
  - **Swap(long element1, long element2)**
    - Swaps two value pair elements, altering their indices.
Click here to see the full Measurement Studio hierarchy chart.
Click on a class above to see the overview for that class.

NOTE: This hierarchy chart may not include all of the classes that are derived from this class. Derived classes that are in different Measurement Studio components are not included.
CNiControl::
WndProc()  ➥

Protected Function

Declared in:
NiControl.h
static long __stdcall WndProc(
    HWND hWnd,
    UINT message,
    WPARAM wParam,
    LPARAM lParam);

Called by the MFC framework to process windows messages.
See Also

Class Overview | CWnd | Hierarchy Chart
Public Constructor

Declared in:
NiControl.h
**Declaration**

```cpp
CNiControl(
    CNiInterface::ThreadAccess threadAccess);
```
**Description**

Default constructor.

**Parameters**

**CNiInterface::ThreadAccess threadAccess**

Specifies how the object can be accessed from multiple threads. The following list includes valid thread access options.

- `CNiInterface::SingleThread`
- `CNiInterface::MultipleThreads`
- `CNiInterface::MultipleThreadsWithCaching`

The thread access specifies the level of multithread support that the object provides. If you do not need to access the object from a thread other than the one that created it, specify `CNiInterface::SingleThread` to optimize performance.

Notes:

1. `CNiInterface::SingleThread` - no multithread support. You can use the object only from the thread in which you created the object or attached the interface pointer.

2. `CNiInterface::MultipleThreads` - full multithread support. You can use the object from any thread. You can destroy the object from any thread.

3. `CNiInterface::MultipleThreadsWithCaching` - full multithread support with caching. You can use the object from any thread. The object internally caches the interface pointer, when possible, to increase performance. A consequence of the caching is that you must destroy the object or detach the interface pointer in the same thread in which you constructed the object or attached the interface pointer.
See Also

- [Class Overview](#)
- [CWnd](#)
- [Hierarchy Chart](#)
Public Destructor

Declared in: NiControl.h
~CNiControl();
**Description**

Destructor.
See Also

- [Class Overview](#)
- [CWnd](#)
- [Hierarchy Chart](#)
CNiControl::
GetCustomInterface()  

Protected Function

Declared in:  
NiControl.h
virtual void * GetCustomInterface(
  bool *pDidAddRef);
Description

Returns the custom interface pointer associated with the object.
See Also

Class Overview | CWnd | Hierarchy Chart
Public Function

GetThreadAccess()  ▲  ◆  ◆  ◆

Declared in:  NiControl.h
Declaration

\texttt{CNiInterface::ThreadAccess GetThreadAccess();}
**Description**

Returns the thread access option to which the control is configured.
See Also

- Class Overview
- CWnd
- Hierarchy Chart
CNiControl:nativeInterface()

Protected Function

Declared in: NiControl.h
virtual void InitCustomInterface();
Description

Initializes the custom interface pointer to associate with the object.
See Also

- Class Overview
- CWnd
- Hierarchy Chart
Protected Function

Declared in:  
NiControl.h
virtual void PreSubclassWindow();
**Description**

Called by the MFC framework before the window is created.

This is a virtual function that you can override in a derived class to add custom behavior. If you override this function, you must call `CNiControl::PreSubclassWindow` in your function definition if you want to allow the library to perform its operations.
See Also

Class Overview | Â« CWnd | Hierarchy Chart
CNiControl::
ReleaseCustomInterface()  

Protected Function

Declared in:
NiControl.h
virtual void ReleaseCustomInterface();
**Description**

Releases the custom interface pointer associated with the object.
See Also

- Class Overview
- CWnd
- Hierarchy Chart
CNiControl::
ValidateControl()  

Protected Function

Declared in: NiControl.h
Declaration

virtual void ValidateControl();
**Description**

Validates the current state of the control.
Click here to see the full Measurement Studio hierarchy chart.

Click on a class above to see the overview for that class.

NOTE: This hierarchy chart may not include all of the classes that are derived from this class. Derived classes that are in different Measurement Studio components are not included.
CNiValuePair:
operator =()  

Public Operator

Declared in:
NiValuepair.h
const CNiValuePair & operator =(
    const CNiValuePair & source);
**Description**

Assignment operator. The object is attached to the same `CWValuePair_CI` pointer as the object to which it is assigned and the object has the same thread access option. The reference count of the `CWValuePair_CI` pointer is incremented. If an `CWValuePair_CI` pointer is already attached to this object, that `CWValuePair_CI` pointer is released and its reference count decremented before the new `CWValuePair_CI` pointer is attached.

**Parameters**

```cpp
cnst CNiValuePair& source
```

Specifies object to which to be assigned.
See Also

Class Overview | CNInterface | Hierarchy Chart
CNiValuePair:

Name  

Public Data Item

Declared in:
NiValuepair.h
Declaration

CString Name;
**Description**

Associated name.
See Also

Class Overview | ▲ CNILInterface | ✏️ Hierarchy Chart
CNIValuePair::

Value

Public Data Item

Declared in:
NiValuepair.h
double Value;
**Description**

Associated value.

Note: See the CNiValuePair overview for information about using date/time values.
See Also

Class Overview | CNInterface | Hierarchy Chart
Public Constructor

Declared in: NiValuepair.h
Declaration

CNiValuePair();
Description

Default constructor. The object constructed by this constructor is in an empty state and has not been connected to a COM object. You should not call the methods and properties of the object until you either assign the object to another CNiValuePair object that is in a usable state or you attach the object to a valid COM object interface pointer.
See Also

- [Class Overview](#)
- [CNiInterface](#)
- [Hierarchy Chart](#)
Public Constructor

Declared in: NiValuepair.h


Declaration

```cpp
CNiValuePair(
    CWValuePair_CI* pCustom,
    CNiInterface::ThreadAccess option);
```
**Description**

Constructor that attaches to the specified CWValuePair_CI pointer.

**Parameters**

CWValuePair_CI* pCustom

Specifies the CWValuePair_CI pointer to which to attach the object.

CNiInterface::ThreadAccess option

Specifies how the object can be accessed from multiple threads. The following list includes valid thread access options.

- CNiInterface::SingleThread
- CNiInterface::MultipleThreads
- CNiInterface::MultipleThreadsWithCaching

The thread access specifies the level of multithread support that the object provides. If you do not need to access the object from a thread thread other than the one that created it, specify CNiInterface::SingleThread to optimize performance.

Notes:

1. CNiInterface::SingleThread - no multithread support. You can use the object only from the thread in which you created the object or attached the interface pointer.

2. CNiInterface::MultipleThreads - full multithread support. You can use the object from any thread. You can destroy the object from any thread.

3. CNiInterface::MultipleThreadsWithCaching - full multithread support with caching. You can use the object from any thread. The object internally caches the interface pointer, when possible, to increase performance. A consequence of the caching is that you must destroy the object or detach the interface pointer in the same thread in which you constructed the object or attached the interface pointer.
Public Constructor

Declared in:
NiValuepair.h
Declaration

CNiValuePair(
    const CNiValuePair & source);
**Description**

Copy constructor. The newly constructed object is attached to the same interface pointer as the object to be copied. The reference count of the interface pointer is incremented.

**Parameters**

`const CNiValuePair& source`

Specifies object to be copied.
CNiValuePair:
~CNiValuePair()  

Public Destructor

Declared in: NiValuepair.h
Declaration

~CNiValuePair();
**Description**

Destructor. If the object is still connected to a COM object interface pointer, the interface pointer is automatically released and the reference count decremented.
See Also

- Class Overview
- CNInterface
- Hierarchy Chart
CNiValuePair:
GetIid() Public Function

Declared in:
NiValuepair.h


```c

Declaration

static const IID & GetIid();
```

Description

Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects. Note: This function is typically for internal use only.
See Also

Class Overview | CNInterface | Hierarchy Chart
Click here to see the full Measurement Studio hierarchy chart.
Click on a class above to see the overview for that class.

NOTE: This hierarchy chart may not include all of the classes that are derived from this class. Derived classes that are in different Measurement Studio components are not included.
Public Function

Swap()  

Declared in:
NiValuepairs.h
void Swap(
    long element1,
    long element2);
Description
Swaps two value pair elements, altering their indices.

Note: This function is useful on CNiSlide and CNiKnob controls in ValuePairsOnly mode.

Parameters

long element1
Specifies the index of a value pair to be swapped.

long element2
Specifies the index of a second value pair to be swapped.
See Also

- Class Overview
- CNiInterface
- Hierarchy Chart
CNiValuePairs::
Count

Public Data Item

Declared in:
NiValuepairs.h
Declaration

short Count;
**Description**

Returns the number of value pairs in the collection.
GridLines

Public Data Item

Declared in:

NiValuepairs.h
Declaration

bool GridLines;
**Description**

Specifies if grid lines are drawn at value pair locations.
See Also

- Class Overview
- CNInterface
- Hierarchy Chart
LabelType

**Public Data Item**

Declared in:
NiValuepairs.h
Declaration

ValuePairLabels LabelType;
**Description**

Specifies the type of labels to draw for the value pairs. The following list includes valid values for this function.

- **CNiValuePairs::LabelNone** - the axis does not draw the value pair.
- **CNiValuePairs::LabelName** - the axis draws the name of the value pairs.
- **CNiValuePairs::LabelValue** - the axis draws the value of the value pair.
CNIValuePairs::

Location

Public Data Item

Declared in:
NiValuepairs.h
Declaration

ValuePairLocations Location;
**Description**

Specifies if value pairs are placed on the axis by their value or by their index. The following list includes valid values for this function.

- **CNiValuePairs::LocationValue** - the axis draws the value pairs at their value on the axis.
- **CNiValuePairs::LocationIndex** - the axis draws the value pairs at their index on the axis.
See Also

Class Overview | CNiInterface | Hierarchy Chart
CNiValuePairs::
MajorTicks

Public Data Item

Declared in:
NiValuepairs.h
Declaration

bool MajorTicks;
**Description**

Specifies if major ticks are placed at the location of the value pairs.
See Also

Class Overview | ▲ CNiInterface | Hierarchy Chart
Public Constructor

CNiValuePairs::

C NiValuePairs()  ▲ ◤ ◤ ◤ ◤ ◤  ⇔  ⇔

Declared in:

NiValuepairs.h
Declaration

CNiValuePairs();
**Description**

Default constructor. The object constructed by this constructor is in an empty state and has not been connected to a COM object. You should not call the methods and properties of the object until you either assign the object to another `CNiValuePairs` object that is in a usable state or you attach the object to a valid COM object interface pointer.
See Also

- Class Overview
- CNilInterface
- Hierarchy Chart
Public Constructor

Declared in: NiValuepairs.h


\textbf{Declaration}

\begin{verbatim}
CNiValuePairs(
    CWValuePairs_CI* pCustom,
    CNiInterface::ThreadAccess option);
\end{verbatim}
Description

Constructor that attaches to the specified CWValuePairs_CI pointer.

Parameters

CWValuePairs_CI* pCustom

Specifies the CWValuePairs_CI pointer to which to attach the object.

CNiInterface::ThreadAccess option

Specifies how the object can be accessed from multiple threads. The following list includes valid thread access options.

- CNiInterface::SingleThread
- CNiInterface::MultipleThreads
- CNiInterface::MultipleThreadsWithCaching

The thread access specifies the level of multithread support that the object provides. If you do not need to access the object from a thread thread other than the one that created it, specify CNiInterface::SingleThread to optimize performance.

Notes:

1. CNiInterface::SingleThread - no multithread support. You can use the object only from the thread in which you created the object or attached the interface pointer.

2. CNiInterface::MultipleThreads - full multithread support. You can use the object from any thread. You can destroy the object from any thread.

3. CNiInterface::MultipleThreadsWithCaching - full multithread support with caching. You can use the object from any thread. The object internally caches the interface pointer, when possible, to increase performance. A consequence of the caching is that you must destroy the object or detach the interface pointer in the same thread in which you constructed the object or attached the interface pointer.
See Also

Class Overview | CNInterface | Hierarchy Chart
CNiValuePairs::
CNiValuePairs()  

Public Constructor

Declared in: NiValuepairs.h
**Declaration**

```cpp
CNiValuePairs(const CNiValuePairs& source);
```
**Description**

Copy constructor. The newly constructed object is attached to the same interface pointer as the object to be copied. The reference count of the interface pointer is incremented.

**Parameters**

`const CNiValuePairs& source`

Specifies object to be copied.
Public Destructor

Declared in:
NiValuepairs.h
Declaration

~CNiValuePairs();
**Description**

Destructor. If the object is still connected to a COM object interface pointer, the interface pointer is automatically released and the reference count decremented.
See Also

- Class Overview
- CNInterface
- Hierarchy Chart
CNiValuePairs::
Add()

Public Function

Declared in:
NiValuepairs.h
Declaration

CNiValuePair Add();
**Description**

Adds a value pair to the collection and returns the new value pair.
See Also

Class Overview | CNiInterface | Hierarchy Chart
Public Function

Declared in: NiValuepairs.h
static const IID & GetIid();
**Description**

Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects. Note: This function is typically for internal use only.
See Also

Class Overview | CNilInterface | Hierarchy Chart
Public Function

Declared in: NiValuepairs.h
**Declaration**

```
CNiValuePair Item(
    const CString& valuePairName);
```
**Description**

Returns the specified value pair from the collection.

**Parameters**

`const CString& valuePairName`

Specifies the name of the value pair to return.
Class CNiValuePairs::
Item()

Public Function

Declared in:
NiValuepairs.h
Declaration

**CNiValuePair** Item(
  long valuePairIndex);
**Description**

Returns the specified value pair from the collection.

**Parameters**

long valuePairIndex

Specifies the one-based index of the value pair to return.
CNiValuePairs::
operator =()

Public Operator

Declared in:
NiValuepairs.h
const CNiValuePairs & operator =(
    const CNiValuePairs & source);
**Description**

Assignment operator. The object is attached to the same CWValuePairs_CI pointer as the object to which it is assigned and the object has the same thread access option. The reference count of the CWValuePairs_CI pointer is incremented. If an CWValuePairs_CI pointer is already attached to this object, that CWValuePairs_CI pointer is released and its reference count decremented before the new CWValuePairs_CI pointer is attached.

**Parameters**

`const CNiValuePairs& source`

Specifies object to which to be assigned.
See Also

Class Overview | ▲ CNilInterface | ▲ Hierarchy Chart
Remove()  

Public Function

Declared in:
NiValuepairs.h


Declaration

```c++
void Remove(
    const CString& valuePairName);
```
**Description**

Removes the specified value pair from the collection.

**Parameters**

`const CString& valuePairName`

Specifies the name of the value pair to remove.
`CNiValuePairs::Remove()`

Public Function

Declared in: `NiValuepairs.h`
void Remove(
    long valuePairIndex);
**Description**

Removes the specified value pair from the collection.

**Parameters**

`long valuePairIndex`

Specifies the one-based index of the value pair to remove.
See Also

Class Overview | CNilInterface | Hierarchy Chart
CNiValuePairs::
RemoveAll()  

Public Function
Declared in:
NiValuepairs.h
Declaration

void RemoveAll();
**Description**

Removes all value pairs from the collection.
See Also

- [Class Overview](#)
- [CNInterface](#)
- [Hierarchy Chart](#)