

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.

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

CNiControl



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

◆ **C o n s t r u c t o r s**

- ◆ **CNiControl**([CniInterface::ThreadAccess](#) threadAccess) Default constructor.

◆ **D e s t r u c t o r s**

◆ `~CNiControl()`

Destructor.

◆ Functions

◆ virtual void *	<u>GetCustomInterface</u> (bool *pDidAddRef)	Returns the custom interface pointer associated with the object.
◆ <u>CNIInterface::ThreadAccess</u> <u>GetThreadAccess</u> ()		Returns the thread access option to which the control is configured.
◆ virtual void	<u>InitCustomInterface</u> ()	Initializes the custom interface pointer to associate with the object.
◆ virtual void	<u>PreSubclassWindow</u> ()	Called by the MFC framework before the window is created.
◆ virtual void	<u>ReleaseCustomInterface</u> ()	Releases the custom interface pointer associated with the object.
◆ virtual void	<u>ValidateControl</u> ()	Validates the current state of the control.
◆ static long __stdcall	<u>WndProc</u> (HWND hWnd,	Called by the

UINT message, WPARAM
wParam, LPARAM lParam)

MFC
framework to
process
windows
messages.

CNiValuePair



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 $1 / 24 \text{ hours} / 60 \text{ minutes}$, which is $1/86400$ or approximately $1.157407\text{e-}5$. 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

◆ Data Items

- ◆ `CString` Name Associated name.
- ◆ `double` Value Associated value.

◆ 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.

◆ **D e s t r u c t o r s**

◆ `~CNiValuePair()`

Destructor.

◆ **F u n c t i o n s**

- ◆ `static const IID & GetIid()` Returns the globally unique identifier (GUID) of the ActiveX interface to which this class connects.
 - ◆ `const CNiValuePair & operator =(
const
CNiValuePair&
source)` Assignment operator.
-

CNiValuePairs



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

- ◆ `short` **Count** Returns the number of value pairs in the collection.
- ◆ `bool` **GridLines** Specifies if grid lines are drawn at value pair locations.
- ◆ `ValuePairLabels` **LabelType** Specifies the type of labels to draw for the value pairs.
- ◆ `ValuePairLocations` **Location** Specifies if value pairs are placed on the axis by their value or by their index.
- ◆ `bool` **MajorTicks** Specifies if major ticks are placed at the location of the value pairs.

◆ 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.

◆ **D e s t r u c t o r s**

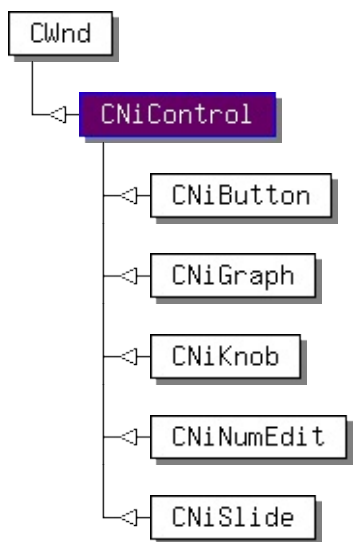
◆ `~CNiValuePairs()`

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.
 - ◆ CNiValuePair **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.
 - ◆ `void` **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.
-

 Hierarchy Chart for:
CNiControl



[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

Declaration

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

Description

Called by the MFC framework to process windows messages.

▸ See Also

 [Class Overview](#) |  [CWnd](#) |  [Hierarchy Chart](#)



CNiControl::

CNiControl()      

Public Constructor

Declared in:
NiControl.h

◆ Declaration

```
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](#)



CNiControl::

`~CNiControl()`      

Public Destructor

Declared in:
NiControl.h

◆ Declaration

```
~CniControl();
```

Description

Destructor.

▸ See Also

 [Class Overview](#) |  [CWnd](#) |  [Hierarchy Chart](#)



CNiControl::

GetCustomInterface()      

Protected Function

Declared in:
NiControl.h

🔑 Declaration

```
virtual void * GetCustomInterface(  
    bool *pDidAddRef);
```

Description







Returns the custom interface pointer associated with the object.

▸ See Also

 [Class Overview](#) |  [CWnd](#) |  [Hierarchy Chart](#)



CNiControl::

GetThreadAccess()      

Public Function

Declared in:
NiControl.h

◆ Declaration

```
CNiInterface::ThreadAccess GetThreadAccess();
```

Description






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

▸ See Also

 [Class Overview](#) |  [CWnd](#) |  [Hierarchy Chart](#)



CNiControl::

InitCustomInterface()      

Protected Function

Declared in:
NiControl.h

Declaration

```
virtual void InitCustomInterface();
```


Description








Initializes the custom interface pointer to associate with the object.

▸ See Also

 [Class Overview](#) |  [CWnd](#) |  [Hierarchy Chart](#)



CNiControl::

PreSubclassWindow()       

Protected Function

Declared in:
NiControl.h

Declaration

```
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](#) |  [CWnd](#) |  [Hierarchy Chart](#)



CNiControl::

ReleaseCustomInterface()       

Protected Function

Declared in:
NiControl.h

Declaration

```
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();
```

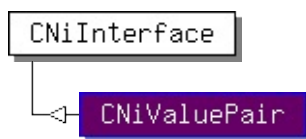
Description

Validates the current state of the control.

▸ See Also

 [Class Overview](#) |  [CWnd](#) |  [Hierarchy Chart](#)

 *Hierarchy Chart for:*
CNiValuePair



[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

◆ Declaration

```
const CNameValuePair & operator =(  
    const CNameValuePair& 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

const CNiValuePair& source

Specifies object to which to be assigned.

▸ See Also

 [Class Overview](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePair::

Name       

Public Data Item

Declared in:
NiValuePair.h

◆ Declaration

`CString` Name;

Description

Associated name.

▸ See Also

 [Class Overview](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePair::

Value        

Public Data Item

Declared in:
NiValuePair.h

◆ Declaration

double Value;

Description

Associated value.

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

▸ See Also

 [Class Overview](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePair::

CNiValuePair()       

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](#)



CNiValuePair::

CNiValuePair()       

Public Constructor

Declared in:
NiValuepair.h

◆ Declaration

```
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 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](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePair::

CNiValuePair()       

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.

▸ See Also

 [Class Overview](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



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](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePair::

Getlid()        

Public Function

Declared in:
NiValuePair.h

◆ Declaration

```
static const IID & GetIid();
```

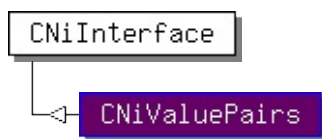
Description

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](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)

 *Hierarchy Chart for:*
CNiValuePairs



[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.



CNiValuePairs::

Swap()        

Public Function

Declared in:
NiValuepairs.h

◆ Declaration

```
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.

▸ See Also

 [Class Overview](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePairs::

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](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePairs::

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.

- `CNValuePairs::LabelNone` - the axis does not draw the value pair.
- `CNValuePairs::LabelName` - the axis draws the name of the value pairs.
- `CNValuePairs::LabelValue` - the axis draws the value of the value pair.

▸ See Also

 [Class Overview](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



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](#)



CNiValuePairs::

CNiValuePairs()       

Public Constructor

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](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePairs::

CNiValuePairs()       

Public Constructor

Declared in:
NiValuepairs.h

◆ Declaration

```
CNiValuePairs(  
    CWValuePairs_CI* pCustom,  
    CNiInterface::ThreadAccess option);
```

☒ 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 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](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePairs::

CNiValuePairs()       

Public Constructor

Declared in:
NiValuepairs.h

◆ Declaration

```
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.

▸ See Also

 [Class Overview](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePairs::

`~CNiValuePairs()`        

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](#) |  [CNIInterface](#) |  [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](#)



CNiValuePairs::

Getlid()       

Public Function

Declared in:
NiValuepairs.h

◆ 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](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePairs::

Item()        

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.

▸ See Also

 [Class Overview](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



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.

▸ See Also

 [Class Overview](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePairs::

operator =()         

Public Operator

Declared in:
NiValuepairs.h

◆ Declaration

```
const CNameValuePair & operator =(  
    const CNameValuePair& source);
```

Description

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

Parameters

const CNiValuePairs& source

Specifies object to which to be assigned.

▸ See Also

 [Class Overview](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePairs::

Remove()       

Public Function

Declared in:
NiValuepairs.h

◆ Declaration

```
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.

▸ See Also

 [Class Overview](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)



CNiValuePairs::

Remove()       

Public Function

Declared in:
NiValuepairs.h

◆ Declaration

```
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](#) |  [CNIInterface](#) |  [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](#) |  [CNIInterface](#) |  [Hierarchy Chart](#)
