ClientAce Help

© 2009 Kepware Technologies

Table of Contents

Help version 1.026

Contents

© Kepware Technologies. Kepware and KEPServerEX are trademarks of Kepware Technologies. Other company and product names mentioned herein are the trademarks or registered trademarks of their respective owners.

ClientAce Overview

ClientAce provides tools to help developers easily build an OPC client application. ClientAce consists of two main parts: the .NET API and the DA Junction.

ClientAce .NET API

The **ClientAce .NET API** (Application Programming Interface) provides C# and Visual Basic .NET language users with a simple, intuitive and optimized class library in order to quickly develop OPC client applications for accessing OPC servers.

ClientAce DA Junction .NET Control

The **ClientAce DA Junction** is a customized .NET control that enables Visual Basic .NET or C# programmers to develop OPC client applications that can access a variety of OPC servers. No detailed knowledge of OPC Data Access interfaces is required. The DA Junction will perform the connection handling procedure between your custom client application and the OPC server, as well as monitoring and reconnecting when necessary. When building advanced custom OPC client applications that require more control over OPC functionality, however, **ClientAce .NET API** is recommended.

Additional ClientAce .NET Controls

ClientAce also includes additional controls that can be used in the Visual Studio Environment. For descriptions and installation instructions, refer to **Additional ClientAce Controls**.

System and Application Requirements

The following requirements must be met in order for the application to operate as designed:

PC Software Requirements
This application supports the following Microsoft Windows operating systems:
- Windows 2003 Server*
- Windows XP*
- Windows 2000 Server
- Windows 2000 Service Pack 2 or higher

*Since Windows Server 2003 and Windows XP have continuous updates, the Windows update feature should be run first to get the latest software.

PC Hardware Requirements
At a minimum, the following hardware is required:
- Intel Pentium III 400 MHz or equivalent processor that supports Microsoft's Windows operating system
- 512 MB installed RAM (256 MB free)
- 40 MB available disk space
- Available Ethernet Card

Microsoft Visual Studio Requirements
ClientAce is currently supported for Microsoft Visual Studio 2003 and Visual Studio 2005. Please note that ASP.NET applications cannot be developed with ClientAce.

.NET Framework Requirements
When deploying custom client applications created using ClientAce, it is required that .NET Framework 1.1 be installed. If the client application utilizes functionality from a version of the .NET Framework that is higher than the .NET 1.1 Framework, that version also will be required on the PC being used to deploy the client. For more information, refer to **Deploying Your Client Application**.

OPC Data Access Requirements
ClientAce supports OPC Data Access (DA) servers that support the following specifications:
- DA server version 2.0
- DA server version 2.05A
- DA server version 3.0

Note: Other DA and OPC servers are not supported at this time.

ClientAce .NET API

Overview of ClientAce .NET API
OpcDaClient Data Model Classes
OpcDaClient Interface of DaServerMgt
OPCCmn Interface of OpcServerEnum Object
OPCCmn ServerIdentifier Class
OPCCmn ServerCategory Enumerator

Overview of .NET Class API

Kepware's ClientAce .NET API provides developers working with languages such as C# and Visual Basic .NET with a simple, intuitive and optimized class library to quickly develop OPC client applications for accessing OPC servers.

Features of the ClientAce .NET API
- A simple, intuitive .NET interface.
- The OPC Data Access interface has been simplified down to the major functions.

- No detailed knowledge of the different OPC Data Access interfaces is required.
- The API covers the different base technologies of OPC, for example, COM and DCOM.
- The API completely covers the connection handling to multiple OPC Servers including connection establishment, connection monitoring and reconnection in case of errors.
- The development of OPC Client applications with C# or Visual Basic .NET becomes very simple using ClientAce.
- Conversion of OPC data from different OPC Data Access interfaces into .NET data types.
- Fast and simple search for OPC COM Servers, both local and remote.
- High performance and optimized Client-Server communication by using kernel functionality implemented in C++.

See Also:
Kepware.ClientAce.OpcDaClient Data Model Classes
Kepware.ClientAce.OpcDaClient Interface of DaServerMgt
Kepware.ClientAce.OPCCmn Interface of OpcServerEnum Object
Licensing ClientAce
Signing Your Client Application

Kepware.ClientAce.OPCCmn ServerIdentifier Class

ServerIdentifier objects are returned by the EnumComServers method and contain information that describe the OPC servers installed on the specified machine.

Public Properties	Type	Description
Category	ServerCategory	Server category (see ServerCategory Enumerator)
CLSID	String	CLSID (Class ID) of the OPC server.
HostName	String	The name or the IP address of the OPC server's host machine (e.g., localhost, PCTest, 192.168.0.120, etc.). If this parameter is left unassigned, the local host is assumed.
ProgID	String	ProgID (program ID) of the OPC server.
Url	String	The url of the server, formatted for use in the **Connect Method**.

Kepware.ClientAce.OPCCmn ServerCategory Enumeration

The ServerCategory enumerator is used to specify the type of OPC server.

Value	Description
OPCAE	Server supports OPC AE 1.10 (alarms and events)
OPCDA	Server supports OPC DA 2.0, 2.05A, and 3.0 (data access)
OPCDX	Server supports OPC DX 1.00 (data exchange)
OPCHDA	Server supports OPC HDA 1.10 (historical data access)
OPCXMLDA	Server supports OPC XMLDA 1.01 (XML data access)

Note: Because OPC XML-DA servers are not registered like COM OPC servers, they cannot be found using the OpcServerEnum object. The URL must be known to connect to an OPC XML-DA server.

Kepware.ClientAce.OpcDaClient Data Model Classes

The DaServerMgt object provides the following functionality in the Kepware.ClientAce.OpcDaClient namespace:

Connection to OPC Server

The Connect method is used to connect to the OPC Server; the Disconnect method is used to release the connection. Because the connection is monitored by ClientAce, the client will be notified of changes in connection status through ServerStateChanged events.

Notification of Data Changes

To avoid cyclic reading, ClientAce API provides tools which notify the client of changes in values. Items can be registered for monitoring by using the Subscribe method; Subscriptions can be cancelled using the SubscriptionCancel method. Notifications of changed values are made by the DataChanged event. Items can be added or removed from a subscription at any time using the SubscriptionAddItems and SubscriptionRemoveItems methods respectively. Subscription properties (such as update rate, active state, and deadband) can also be changed at any time using the SubscriptionModify method.

Read and Write of OPC Data Access Items

The values of OPC items can be changed using the asynchronous WriteAsync and synchronous Write methods. The values can be obtained when subscription is not appropriate by using the asynchronous ReadAsync and synchronous Read methods.

Obtaining Information on the Address Space

The Address Space Browse method can be used to search for OPC items. The GetProperties method can be used to obtain the properties of OPC items.

DaServerMgt Class

The DaServerMgt class allows access to an OPC Data Access Server. For a more detailed description of the ClientAce API and its methods, refer to **Kepware.ClientAce.OpcDaClient Interface of DaServerMgt**, beginning with **Creating DaServerMgt Object**.

ServerState Enumeration

Changes in server connection state, as indicated in ServerStateChanged events, may have one of the following enumerated values:

Value	Description
CONNECTED	The server is connected.
DISCONNECTED	The server is disconnected.
ERRORSHUTDOWN	The server is shutting down.
ERRORWATCHDOG	The ClientAce API watchdog has determined that a server connection has failed. ClientAce may attempt to reconnect to the server depending on the options specified when the Connect method was called.
UNDEFINED	The server state is not known.

ItemIdentifier Class

The ItemIdentifier class is a required parameter of the following methods:

- GetProperties
- Read
- ReadAsync
- Subscribe
- SubscriptionAddItems
- SubscriptionRemoveItems
- Write
- WriteAsync

ItemIdentifier objects are used to identify OPC items within a server. These objects are passed by reference (in/out) in all method calls so that ClientAce may update certain properties as described below.

Public Properties	Type	Description
ClientHandle	Object	ClientAce will reference items in DataChanged, ReadCompleted, and WriteCompleted events by their ClientHandle. A handle can be assigned to access the data

		storage object for the item. This storage object could be a TextBox control on the GUI or an instance of a custom class defined in the application. (See provided Simple and Complex examples installed with ClientAce).
DataType	System.Type	When an ItemItentifier object is first used, the property may be used to specify the data type which the item value will be received as. If the server cannot provide the requested type for this item, ClientAce will indicate this through the ResultID and reset this property to the item's Native, or canonical (default) data type. If this property is left unspecified, ClientAce will reset this property with the item's canonical (default) data type.
ItemName	String	This property contains the name (ItemID) of an OPC Data Access item.
ItemPath	String	Reserved for future use.
ResultID	ResultID	Whenever an item specific error occurs during and OPC call (such as, unknown ItemName, trying to write to a read only item, unsupported data type, etc.), the error code provided by the server will be placed in the ResultID object for the associated ItemItendifier. ClientAce will provide additional descriptive information for the error. If a ClientAce API call returns a **ReturnCode** indicating an error, the ResultID of all ItemIdentifiers passed to the method should be examined to see which items failed and why.
ServerHandle	Integer	The API will set this value when the ItemIdentifier is first used. The API can use the ServerHandle to optimize future calls to the OPC server.

ItemValue Class

The ItemValue class is used in the following methods:
- Read
- Write
- WriteAsync

The **ItemValue** contains the value, quality and time stamp of an OPC item.
The **Read** method takes an array of ItemValue objects as an output parameter.
The **API** allocates and fills the array with the requested item values during the read.
The **Write** and **WriteAsync** methods takes an array of ItemValue objects as an input parameter. This array must be filled with the values to be written to the items specified in the corresponding array of ItemIdentifier objects.

Public Properties	Type	Description
Quality	QualityID (see **Class QualityID**)	The OPC quality of the associated Value. The class QualityID provides the quality code (int), the name (string) and the description (string). This value is Read Only and is set by the API during reads.
TimeStamp	Date	The time stamp of the associated Value. This value is Read Only and is set by the API during reads.
Value	Object	The value of the item. Being an object, it can contain any data type. Typically the value will be of the same type as requested by the corresponding ItemIdentifier. If no type was specified, the value will be provided in its canonical form.

ItemValueCallback Class

ItemValueCallback is derived from the ItemValue class and is used in DataChanged and ReadCompleted events.
ItemValueCallback objects will have the following properties:

Public Properties	Type	Description
ClientHandle	Object	This is the client handle of the item specified in the call to Subscribe or ReadAsync. The client uses this handle to access the appropriate storage object for the received data.
Quality	QualityID (see **Class QualityID**)	The quality associated with the value when it was acquired from the data source. The class QualityID provides the quality code (int), the name (string) and the description (string). This value is Read Only and is set by the API during reads.
ResultID	ResultID (see **Class ResultID**)	The class ResultID provides the error code (int), the name (string) and a language dependant description (string) for the item represented by the ClientHandle. Thus certain activity can be programmed to react on eventually occurring errors. It is also possible to simply display the error on the user interface (message box).
TimeStamp	Date	The time stamp of the associated Value. This value is Read Only and is set by the API during reads.
Value	Object	The value of the item. Being an object, it can contain any data type. Typically the Value will be of the same type as requested by the corresponding ItemIdentifier. If no type was specified, the value will be provided in its canonical form.

Note: Quality, TimeStamp and Value are shared from the base class.

ItemResultCallback Class

The ItemResultCallback class is used in the WriteCompleted event.

Public Properties	Type	Description
ClientHandle	Object	This is the client handle of the item specified in the call to WriteAsync. The client uses this handle to access the appropriate storage object for the received data.
ResultID	ResultID (see **Class ResultID**)	The class ResultID provides the error code (int), the name (string) and a language dependant description (string) for the item represented by the ClientHandle. Thus certain activity can be programmed to react on eventually occurring errors. It is also possible to simply display the error on the user interface (i.e., the message box).

BrowseElement Class

The BrowseElement class contains all the information that was obtained by using the Browse method.

Public Properties	Type	Description
HasChildren	Boolean	True if the element has child elements in the address space, otherwise false.
IsItem	Boolean	True if the element is an OPC Data Access item, otherwise false.
ItemName	String	The item name of the element.
ItemPath	String	The item path of the element.
ItemProperties	ItemProperties (see **ClassItemProperties**)	The properties of the element that were available through Browse method.
Name	String	The name of the returned element. Typically this name is used for displaying the address space in a tree or other structured format.

BrowseFilter Enumeration

The BrowseFilter Enumeration is used to specify the type of child elements returned by the Browse method. Possible filters are as follows:

Value	Description
ALL	All elements will be returned.
BRANCH	Only elements of type Branch will be returned.
ITEM	Only elements of type Item will be returned.

ItemProperties Class

Objects of this class will be returned by the Browse and GetProperties methods, and will contain all of the requested properties of a single OPC item.

Public Properties	Type	Description
RequestedItemProperties	ItemProperty (see **Class ItemProperty**)	Array of objects of class ItemProperty. This array contains all requested properties of an OPC Item.

ItemProperty Class

ItemProperty objects are used to describe a single property of an OPC item.

Public Properties	Type	Description
DataType	System.Type	The data type of the property value.
Description	String	The description of the property. This information can be used when displaying the property in a graphical user interface, such as in a Grid Control or a ToolTip).
ItemName	String	If the OPC Server supports reading and writing of properties through an item, here the item name of this property will be returned.
ItemPath	String	If the OPC Server supports reading and writing of properties through an item, here the item path of this property will be returned.
PropertyID	Integer	The identification number of the property.
ResultID	ResultID (see **Class ResultID**)	If an error occurred while obtaining the properties, the dedicated error code will be returned within this object.
Value	Object	The value of the property.

ResultID Class

ResultID objects are used to describe the result of an operation on an OPC item, such as read, write, subscribe. ResultID objects will contain the error code provided by the server, its string representation and a description of the error code. Each item will have its own ResultIDm since requests that contain multiple items may succeed for some items and fail for other items.

Public Properties	Type	Description
Code	Integer	The code sent by the server for the particular action.
Description	String	The description of the error (language depends on the locale).
Name	String	The string representation of the code.
Succeeded	Boolean	This property will be True if the operation was a success for the item, or False if it failed. If this is False, the specific reason for failure can be determined by examining the other properties.

QualityID Class

A QualityID object is used to describe the OPC quality of an item's value.

Public Properties	Type	Description
Description	String	Description of the quality code (language depends on the locale).
FullCode	Integer	The full code sent by the server.
IsGood	Boolean	This property will be True if the value has "good" quality. If False, detailed information about the quality of the value can be determined from the other properties.
LimitBits	Integer	The limit portion of the code sent by the server. Please see **Appendix 3** for a full discussion of OPC Quality based on the OPC specifications.
Name	String	String representation of the code. Please see **Appendix 3** for a full discussion of OPC Quality based on the OPC specifications.
Quality	Integer	Code that indicates the quality of the value sent by the server. Please see **Appendix 3** for a full discussion of OPC Quality based on the OPC specifications.
VendorBits	Integer	Vendor-specific data within the code. Please see **Appendix 3** for a full discussion of OPC Quality based on the OPC specifications.

ConnectInfo Class

A ConnectInfo object is used to pass connection related options to the API. This information determines how the API will monitor and maintain connections, and also provide language dependent strings.

Public Properties	Type	Description
KeepAliveTime	Integer	During runtime the API continuously checks the availability of the connection to the server. KeepAliveTime represents the time interval, in milliseconds, at which this availability check takes place. The default value is 10,000 ms. The API will start reconnection attempts at an interval of two times KeepAliveTime and will be incremented by 1 KeepAliveTime up to 10 times KeepAliveTime if the server is not available for a longer time period. The reconnect interval after a shutdown event from the OPC server is one minute. For example, if KeepAliveTime = 10,000 ms, then the first reconnect attempt will be 20 seconds after check-fail; the second reconnect attempt will be 30 seconds after the first; the third reconnect attempt will be 40 seconds after the second, and so on up to 100 seconds. From that point on, retries will continue every 100 seconds.
LocalID	String	Using LocalID, a country abbreviation (en-us, en, etc.) can be passed to the server. When the LocalID is set, the language-dependent return values will be returned in the selected language, if supported by the OPC server. If the value cannot be found, the default value will be passed to the server.
RetryAfterConnectionError	Boolean	If this flag is set, the API will attempt to reconnect after a connection loss until the reconnect succeeds. If the connection can be re-established, all handles that were created before the connection loss will be valid again. Event handler methods will not have to be re-registered.
RetryInitialConnection	Boolean	If this flag is set to true, the API will try to connect to the server even when the first connect did not succeed.

Note: Changes in the connection status should be monitored using a ServerStateChanged event handler. Connect is the only method in the DaServerMgt namespace that can be called prior to establishing a connection. This can be tested at any time with the **IsConnected property**.

ReturnCode Enumeration

Most ClientAce API methods will return a code indicating the level of success of the operation. The code may take one of the following enumerated values. In the event that the function cannot satisfy the request due to invalid arguments or unexpected errors, an exception will be thrown.

Value	Description
ITEMANDQUALITYERROR	An error was returned during operation for at least one item. The returned quality for at least one item (either the same or different item) was not good. The items can be determined by checking the ResultID and the quality field of the ItemIdentifier array.
ITEMERROR	For at least one item, an error was returned during operation. The item can be determined by checking the ResultID of the ItemIdentifier array.
QUALITYNOTGOOD	For at least one item, the returned quality was not good. The item can be determined by checking the quality field of the ItemIdentifier array.
SUCCEEDED	The function returned successfully.
UNSUPPORTEDUPDATERATE	The function returned successfully, but the requested update was not supported by the underlying server. The revised update will be returned to the client (Subscribe and SubscriptionModify methods only).

Kepware.ClientAce.OpcDaClient Interface of DaServerMgt

Creating DaServerMgt Object

The first step is to create an instance of DaServerMgt.

```
[Visual Basic]
Dim WithEvents daServerMgt As New Kepware.ClientAce.OpcDaClient.DAServerMgt
```

```
[C#]

DaServerMgt daServerMgt = new Kepware.ClientAce.OpcDaClient.DaServerMgt ();
```

Connect Method

```
[Visual Basic]

Connect ( _

        ByVal url As String, _

        ByVal clientHandle As Integer, _

        ByRef connectInfo As Kepware.ClientAce.OpcDaClient.ConnectInfo, _

        ByRef connectFailed As Boolean _

)
```

```
[C#]

void Connect (

        string url,

        int clientHandle

        ref Kepware.ClientAce.OpcDaClient.ConnectInfo connectInfo,

        out bool connectFailed

);
```

The Connect method establishes a connection with an OPC server.

Parameter	Functionalities
url	The URL of the OPC servers. **Note:** The syntax of the URL that uniquely identifies a server must follow this format: [OpcSpecification]://[Hostname]/[ServerIdentifier] OpcSpecification: Selects the OPC Specification to be used. • opcda for OPC Data Access 2.05A respectively 3.0 (COM)Hostname: Name or IP address of the machine that hosts the OPC server. For the local machine, localhost must be used. ServerIdentifier: Identifies the OPC server on the specified host. • OPC COM DA – [ProgID]/[optional ClassID] **Note:** For OPC DA servers, the API will attempt to connect using the ClassID first. If the ClassID is not given, or is found to be invalid, the API will attempt to connect using the ProgID.

	Examples:
	opcda://localhost/OPCSample.OpcDaServer/{625c49a1-be1c-45d7-9a8a-14bedcf5ce6c}
	opcda://PC_001/ KEPware.KEPServerEx.V4/{6e6170f0-ff2d-11d2-8087-00105aa8f840}
	opcda://PC_001/ KEPware.KEPServerEx.V4
	opcda://PC_001//{6e6170f0-ff2d-11d2-8087-00105aa8f840}
clientHandle	The client application can specify a handle to uniquely identify a server connection. The API will return this handle in ServerStateChanged events.
connectInfo	Additional connection options are specified using the connectInfo parameter. See **Class ConnectInfo** for more information.
	Indicates whether or not the initial connection to the underlying server failed. This setting only applies if the retryConnect flag was set in the connect call.

Examples

```vbnet
[ Visual Basic]

' Declare variables

Dim url As String = "opcda://localhost/KEPware.OPCSampleServer/{6E617113-FF2D-11D2-8087-00105AA8F840}"

Dim clientHandle As Integer = 1

Dim connectInfo As New Kepware.ClientAce.OpcDaClient.ConnectInfo

connectInfo.LocalID = "en"

connectInfo.KeepAliveTime = 5000

connectInfo.RetryAfterConnectionError = True

connectInfo.RetryInitialConnection = True

Dim connectFailed As Boolean

Try

    ' Call Connect API method

    daServerMgt.Connect( _

        url, _

        clientHandle, _

        connectInfo, _

        connectFailed)

    ' Check result

    If connectFailed = True Then

    Console.WriteLine("Connect failed.")

    End If

Catch ex As Exception
```

```
    Console.WriteLine("Connect exception. Reason: " & ex.Message)

End Try
```

```
[C#]

// Declare variables

string url = "opcda://localhost/KEPware.OPCSampleServer/{6E617113-FF2D-11D2-8087-00105AA8F840}";

int clientHandle = 1;

ConnectInfo connectInfo = new ConnectInfo();

connectInfo.LocalID = "en";

connectInfo.KeepAliveTime = 5000;

connectInfo.RetryAfterConnectionError = true;

connectInfo.RetryInitialConnection = true;

bool connectFailed;

try

{

     // Call Connect API method

    daServerMgt.Connect(url, clientHandle, ref connectInfo, out connectFailed);

    // Check result

    if (connectFailed)

    {

            Console.WriteLine("Connect failed.");

    }

}

catch (Exception ex)

{

     Console.WriteLine("Connect exception. Reason: {0}", ex);

}
```

Note 1: The IsConnected property indicates that a client application has successfully called the Connect method. This does not necessarily indicate whether ClientAce is connected to the server. For example: This property would remain true after a connection has failed and ClientAce is in the process of reconnecting. To test the ClientAce to server connection state, use the **ServerState property**. The server connection state may also be monitored by implementing the **ServerStateChanged** event handler.

Note 2: It is highly recommended that client applications wait at least 1 second after disconnecting from a server before attempting to connect to that server again.

Disconnect Method

```
[Visual Basic]

Disconnect ()
```

```
[C#]

void Disconnect ();
```

Note: By calling the Disconnect method, the connection to the OPC Server is released. All subscriptions and resources will be freed.

Examples

```
[Visual Basic]

If daServerMgt.IsConnected = True Then

    daServerMgt.Disconnect()

End If
```

```
[C#]


if (daServerMgt.IsConnected)

        daServerMgt.Disconnect();
```

IsConnected Property

```
[Visual Basic

IsConnected () As Boolean
```

```
[C#]

bool IsConnected ();
```

Note: This property is used to check if the client application has successfully called the Connect method. Possible return values are:

Value	Description
True	The client is connected to ClientAce
False	The client is not connected to ClientAce

Note: The IsConnected property indicates that a client application has successfully called the Connect method. It does not necessarily indicate whether ClientAce is connected to the server. For example: Such a property would remain true even after a connection has failed and ClientAce is in the process of reconnecting. To test the ClientAce to server connection state, use the **ServerState Property**. To monitor the server connection state, implement the **ServerStateChanged event handler**.

ServerState Property

```
[ Visual Basic]

ServerState () As Kepware.ClientAce.OpcDaClient.ServerState
```

```
[ C#]

Kepware.ClientAce.OpcDaClient.ServerState ServerState();
```

Use ServerState, not the IsConnected property, to determine the status of the server connection. Parameters:

Value	Description
ServerState (see **Enumerator ServerState**)	Describes the current connection state between the ClientAce API and the OPC server.

Browse Method

```
[ Visual Basic]

Browse ( _

      ByVal itemName As String, _

      ByVal itemPath As String, _

      ByRef continuationPoint  As String, _

      ByVal maxElementsReturned As Integer, _

      ByVal browseFilter As Kepware.ClientAce.OpcDaClient.BrowseFilter, _

      ByVal propertyIDs() As Integer, _

      ByVal returnAllProperties As Boolean, _

      ByVal returnPropertyValues As Boolean, _

      ByRef browseElements() As Kepware.ClientAce.OpcDaClient.BrowseElement, _

      ByRef moreElements As Boolean _

) As Kepware.ClientAce.OpcDaClient.ReturnCode
```

```
[ C#]

As Kepware.ClientAce.OpcDaClient.ReturnCode Browse (

      string itemName,

      string itemPath,

      ref string continuationPoint,

      int maxElementsReturned,

      Kepware.ClientAce.OpcDaClient.BrowseFilter browseFilter,
```

```
    int[] propertyIDs,

    bool returnAllProperties,

    bool returnPropertyValues,

    out Kepware.ClientAce.OpcDaClient.BrowseElement[] browseElements,

    out bool moreElements

);
```

The Browse method is used to search for tags in the address space of an OPC Server. The address space is usually displayed in a tree structure because it is close to the outline of the items and branches of the internal hierarchical structure of the server itself.

Parameter	Functionality
itemName	This parameter specifies the element (branch) for which all child elements will be obtained. If an empty string is passed, the root level of the server will be browsed.
itemPath	Reserved for future use.
continuationPoint	If the number of returned elements is limited by the client (parameter maxElementsReturned) or if the server limits the returned elements to a certain number, this parameter is provided to specify a reference point for follow up Browse calls regarding this element in the server's hierarchy. If an OPC server returns a continuation point, the Browse must be called again with the same parameters but using the returned Continuation Point to obtain missing child elements of this node.
maxElementsReturned	This parameter can be used to define the maximum number of elements the server should return. If this value is set to 0, **all** elements will be returned.
browseFilter	The BrowseFilter is used to define the type of elements to be returned. Possible values are **all**, **items** or **branches**
propertyIDs	This parameter is used to specify the properties that should be obtained when calling the Browse. The properties will be returned in the associated BrowseElement. This will be ignored if the returnAllProperties parameter is set to True.
returnAllProperties	If the returnAllProperties flag is set to true, all properties of the items will be obtained automatically. The properties will be returned in the associated BrowseElement.
returnPropertyValues	If the returnPropertyValues flag is set to true, the values of the requested properties will be returned.
browseElements	This array contains all child elements of the element specified in ItemName.
moreElements	The moreElements parameter indicates when not all child elements are returned.

Note 1: For more information on Return Value: ReturnCode, refer to **ReturnCode Enumerator**. In the event that the function cannot satisfy the request due to invalid arguments or unexpected errors, an exception will be thrown.

Note 2 :
Before the Browse method is called, its parent DaServerMgt object must be connected to an OPC server using the Connect method. Otherwise, a null reference exception will be thrown.

Examples

This example shows how to browse the entire namespace of the connected server using recursive functions calls. The results are placed in a tree view control named **tvItems**.

```
[Visual Basic]

' Create root node

tvItems.Nodes.Add("KepServerEx")
```

```vb
Dim rootNode As TreeNode = tvItems.Nodes(0)

' Browse from root

Browse("", rootNode)

' Additional code

Private Sub Browse(ByVal branchName As String, ByVal node As TreeNode)

    Dim itemName As String

    Dim itemPath As String

    Dim continuationPoint As String = ""

    Dim maxElementsReturned As Integer

    Dim browseFilter As Kepware.ClientAce.OpcDaClient.BrowseFilter

    Dim propertyIDs() As Integer

    Dim returnAllProperties As Boolean

    Dim returnPropertyValues As Boolean

    Dim browseElements() As Kepware.ClientAce.OpcDaClient.BrowseElement

    Dim moreElements As Boolean = True

    ' Set input parameters

    itemName = branchName

    itemPath = ""

    maxElementsReturned = 0

    browseFilter = Kepware.ClientAce.OpcDaClient.BrowseFilter.ALL

    propertyIDs = Nothing ' prevent Visual Studio warning

    returnAllProperties = True

    returnPropertyValues = False

    browseElements = Nothing ' prevent Visual Studio warning

    ' Call Browse API method
```

(Continued)

(VB example continuation)

```vb
    Try

        While moreElements = True

            daServerMgt.Browse( itemName, _
```

```vb
                    itemPath, _

                    continuationPoint, _

                    maxElementsReturned, _

                    browseFilter, _

                    propertyIDs, _

                    returnAllProperties, _

                    returnPropertyValues, _

                    browseElements, _

                    moreElements)

         ' Handle results

         Dim numberOfElementsReturned As Integer = _browseElements.GetLength(0)

         Dim element As Integer

         For element = 0 To numberOfElementsReturned - 1

                 ' Add item to specified tree node

                 node.Nodes.Add(browseElements(element).Name)

                 ' Browse for item's children (recursive call!!!)

                 If browseElements(element).HasChildren Then

                    itemName = browseElements(element).ItemName

                    Browse( browseElements(element).ItemName, node.Nodes(element))

                 End If

            Next

        End While

    Catch ex As Exception

        MsgBox("Browse exception: " & ex.Message)

    End Try

End Sub
```

```csharp
[C#]

// Create root node

tvItems.Nodes.Add("KepServerEx");

TreeNode rootNode = tvItems.Nodes[0];
```

```
// Browse from root

Browse("", rootNode);

// Additional code

private void Browse(string branchName, TreeNode node)

{

// Declare parameters

      string itemName;

      string itemPath;

      string continuationPoint = "";

      int maxElementsReturned;

      BrowseFilter browseFilter;

      int[] propertyIDs = null;

      bool returnAllProperties;

      bool returnPropertyValues;

      BrowseElement[] browseElements = null;

      bool moreElements = true;

      // Set input parameters

      itemName = branchName;

      itemPath = "";

      maxElementsReturned = 0;

      browseFilter = BrowseFilter.ALL;

      returnAllProperties = true;

      returnPropertyValues = false;
```

(Continued)

(C# example continuation)

```
      // Call Browse API method

      try

      {

          while (moreElements == true)

          {
```

```
   daServerMgt.Browse(itemName, itemPath, ref continuationPoint,

maxElementsReturned, browseFilter, propertyIDs,

returnAllProperties, returnPropertyValues, out browseElements, out

moreElements);

            // Handle results

            int numberOfElementsReturned = browseElements.GetLength(0);

            int element;

            for (element = 0; element < numberOfElementsReturned; element++)

            {

                // Add item to specified tree node

                node.Nodes.Add(browseElements[element].Name);

                // Browse for item's children (recursive call!!!)

                if (browseElements[element].HasChildren)

                {

                itemName = browseElements[element].ItemName;

                Browse(browseElements[element].ItemName, node.Nodes[element]);

                }

            }

        }

    catch (Exception ex)

    {

        Console.WriteLine("Browse exception. Reason: {0}", ex);

    }

}
```

Get Properties Method

```
[Visual Basic]

GetProperties ( _

    ByRef itemIdentifiers As Kepware.ClientAce.OpcDaClient.ItemIdentifier, _
```

```
      ByVal propertyIDs() As Integer, _

      ByVal returnAllProperties As Boolean, _

      ByVal returnPropertyValues As Boolean, _

      ByRef itemProperties() As Kepware.ClientAce.OpcDaClient.ItemProperties, _

) As Kepware.ClientAce.OpcDaClient.ReturnCode
```

```csharp
[C#]

Kepware.ClientAce.OpcDaClient.ReturnCode GetProperties (

      ref Kepware.ClientAce.OpcDaClient.ItemIdentifier[] itemIdentifiers,

      int[] propertyIDs,

      bool returnAllProperties,

      bool returnPropertyValues,

      out Kepware.ClientAce.OpcDaClient.ItemProperties[] itemProperties

);
```

Note: The GetProperties method is used to obtain the properties of OPC items.

Parameter	Functionality
itemIdentifiers	The array of itemIdentifiers is used to specify the OPC items you which to obtain the properties of.
propertyIDs	The IDs of the properties to be obtained by the GetProperties call. The properties will be returned in the associated itemProperties element. This will be ignored if the returnAllProperties parameter is set to True.
returnAllProperties	If this flag is set to True, all properties of the items will be obtained automatically. The properties will be returned in the associated itemProperties element.
returnPropertyValues	The property values will be returned if this flag is set to True.
itemProperties	This array contains ItemProperty objects describing the requested properties of the items.

Note: For more information on Return Value: ReturnCode, refer to **ReturnCode Enumerator**. In the event that the function cannot satisfy the request due to invalid arguments or unexpected errors, an exception will be thrown.

Examples
This example shows how to get the access rights and data type properties of a single item **Channel_1.Device_1. Tag_1**.

```vbnet
[Visual Basic]

' Declare variables

Dim itemIdentifiers(0) As Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_1"

Dim propertyIDs(1) As Integer
```

```vb
propertyIDs(0) = Kepware.ClientAce.OpcDaClient.PropertyID.ACCESSRIGHTS

propertyIDs(1) = Kepware.ClientAce.OpcDaClient.PropertyID.DATATYPE

Dim returnAllProperties As Boolean = False

Dim returnPropertyValues As Boolean = True

Dim itemProperties() As Kepware.ClientAce.OpcDaClient.ItemProperties

Try

    ' Call GetProperties API method

    daServerMgt.GetProperties( _

    itemIdentifiers, _

    propertyIDs, _

    returnAllProperties, _

    returnPropertyValues, _

    itemProperties)

    ' Handle results

    Dim itemProperty As Kepware.ClientAce.OpcDaClient.ItemProperty

    For Each itemProperty In itemProperties(0).RequestedItemProperties

        Dim propertyDescription As String = itemProperty.Description()

        Dim propertyValue As String = itemProperty.Value.ToString()

        Console.WriteLine( _

            "Property: " & propertyDescription & _

            " Value: " & propertyValue)

    Next

Catch ex As Exception

    Console.WriteLine("GetProperties exception. Reason: " & ex.Message)

    End Try
```

```csharp
[C#]

// Declare variables

ItemIdentifier[] itemIdentifiers = new ItemIdentifier[1];

itemIdentifiers[0] = new ItemIdentifier();

itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_1";
```

```
int[] propertyIDs = new int[2];

propertyIDs[0] = (int)PropertyID.ACCESSRIGHTS;

propertyIDs[1] = (int)PropertyID.DATATYPE;

bool returnAllProperties = false;

bool returnPropertyValues = true;

ItemProperties[] itemProperties = null;

try

{

      // Call GetProperties API method

          daServerMgt.GetProperties(ref itemIdentifiers, propertyIDs,
          returnAllProperties, returnPropertyValues, out itemProperties);

      // Handle results

           foreach (ItemProperty itemProperty in itemProperties[0].
           RequestedItemProperties)

      {

              string propertyDescription = itemProperty.Description;

              string propertyValue = itemProperty.Value.ToString();

              Console.WriteLine("Property: {0} Value: {1}",

                        propertyDescription,

                        propertyValue);

      }

}

    catch (Exception ex)

{

      Console.WriteLine("GetProperties exception. Reason: {0}", ex);

}
```

Subscribe Method

```
[Visual Basic]

Subscribe ( _

      ByVal clientSubscription As Integer, _
```

```
      ByVal active As Boolean, _

      ByVal updateRate As Integer, _

      ByRef revisedUpdateRate As Integer, _

      ByVal deadband As Single, _

      ByRef itemIdentifiers() As Kepware.ClientAce.OpcDaClient.ItemIdentifier, _

      ByRef serverSubscription As Integer _

) As Kepware.ClientAce.OpcDaClient.ReturnCode
```

```
[C#]

Kepware.ClientAce.OpcDaClient.ReturnCode Subscribe (

      int clientSubscription,

      bool active,

      int updateRate,

      out int revisedUpdateRate,

      float deadband,

      ref Kepware.ClientAce.OpcDaClient.ItemIdentifier[] itemIdentifiers,

      out int serverSubscription

);
```

The Subscribe method is used to register items for monitoring. The server will continuously scan the subscribed items at the specified update rate and notify the ClientAce API when any item's values or quality changes. The ClientAce API will relay this information to the client application via **DataChanged events**. This relieves the client of having to make continuous calls to Read or ReadAsync to poll a set of items and can greatly improve the performance of the client application and server.

Parameter	Functionality
clientSubscription	With this parameter, a meaningful handle may be assigned to each subscription. This value will be returned in each DataChanged event and provides a means of indicating which subscription the data update is for.
active	This parameter is used to create the subscription as active or inactive. The server will scan the items in a subscription only when the subscription is active. The active state may be changed at any time with the **SubscriptionModify Method**. The subscription active state can be used to optimize the application, by signaling the server to stop scanning items that are not currently of interest.
updateRate	With this parameter, the rate at which the server scans the subscribed items can be specified. This is a requested rate - the actual update rate will be decided by the server at the time of this call, but can still vary depending on demands on the server and data source. Update rate values must be in milliseconds.
revisedUpdateRate	This out parameter returns the update rate set by the OPC server, which can be different from the requested updateRate. The revised update rate will be in milliseconds.
deadband	The deadband parameter specifies the minimum deviation needed for the server to notify the client of a change of value. The deadband is given a percent (0.0–100.0) of the range of the value. The range is given by the EU Low and EU High properties of the item. A deadband of 0.0 will result in the server notifying the client of all changes in the item's

	value. The Subscribe method will throw an exception if an invalid deadband value is specified.
itemIdentifiers	The array of itemIdentifiers is used to specify the OPC items that should be added to the subscription.
serverSubscription	The API will assign a unique handle for each subscription. This handle is returned through this parameter and should be stored for later use. The server subscription handle must be specified when modifying or canceling a subscription.

Note 1: The return code indicates the overall success of the call. If this code indicates an item-specific error (ITEMERROR), each of the ReturnID objects should be examined in order to determine which items could not be added to the subscription and why. The return code will also indicate if the requested update rate is not supported by the server. In the event that the function cannot satisfy the request (due to invalid arguments or unexpected errors), an exception will be thrown. For more information on Return Value:Return Code, refer to **ReturnCode Enumerator**.

Note 2: The server will send an initial update for all items added to an active subscription.

Note 3: In order for the server to return item values with a particular data type, that particular type must be requested with the ItemIdentifier.DataType property. The ResultID will indicate if the server is able to provide the value as the requested type. If the requested type cannot be provided, the values will be sent in their canonical (default) data type.

Examples
This example show how to create a new subscription for the two items **Channel_1.Device_1.Tag_1** and **Channel_1. Device_1.Tag_2**.

```
[Visual Basic]

' Declare variables

Dim clientSubscription As Integer = 1

Dim active As Boolean = True

Dim updateRate As Integer = 500

Dim revisedUpdateRate As Integer

Dim deadband As Single = 0

Dim itemIdentifiers(1) As Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_1"

itemIdentifiers(0).ClientHandle = 1 ' Assign unique handle

itemIdentifiers(1) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(1).ItemName = "Channel_1.Device_1.Tag_2"

itemIdentifiers(1).ClientHandle = 2 ' Assign unique handle

Dim serverSubscription As Integer

Try

   ' Call Subscribe API method

   daServerMgt.Subscribe( _
```

```vb
        clientSubscription, active, updateRate, _

        revisedUpdateRate, deadband, itemIdentifiers, serverSubscription)

    ' Check results

    Dim item As Kepware.ClientAce.OpcDaClient.ItemIdentifier

    For Each item In itemIdentifiers

        If item.ResultID.Succeeded = False Then

            Console.WriteLine("Subscribe failed for item: " & item.ItemName)

        End If

    Next

Catch ex As Exception

    Console.WriteLine("Subscribe exception.  Reaseon: " & ex.Message)

End Try
```

```csharp
[C#]

// Declare variables

int clientSubscription = 1;

bool active = true;

int updateRate = 500;

int revisedUpdateRate;

float deadband = 0;

ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];

itemIdentifiers[0] = new ItemIdentifier();

itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_1";

itemIdentifiers[0].ClientHandle = 1; // Assign unique handle

itemIdentifiers[1] = new ItemIdentifier();

itemIdentifiers[1].ItemName = "Channel_1.Device_1.Tag_2";

itemIdentifiers[1].ClientHandle = 2; // Assign unique handle

int serverSubscription;

ReturnCode returnCode;

try

{  // Call Subscribe API method
```

```
returnCode = daServerMgt.Subscribe(clientSubscription, active, updateRate, out
revisedUpdateRate, deadband, ref itemIdentifiers, out serverSubscription);


     // Check results if (returnCode != ReturnCode.SUCCEEDED)

   { foreach (ItemIdentifier item in itemIdentifiers)

         { if (!item.ResultID.Succeeded)

      { Console.WriteLine("Subscribe failed for item {0}", item.ItemName);

              }

          }

      }

}catch (Exception ex)

{Console.WriteLine("Subscribe exception. Reason: {0}", ex);

}
```

SubscriptionModify Method

```vbnet
[Visual Basic]

SubscriptionModify ( _

      ByVal serverSubscription As Integer, _

      ByVal active As Boolean, _

      ByVal updateRate As Integer, _

      ByRef revisedUpdateRate As Integer, _

      ByVal deadband As Single _

) Kepware.ClientAce.OpcDaClient.ReturnCode

SubscriptionModify ( _

      ByVal serverSubscription As Integer, _

      ByVal active As Boolean _

) Kepware.ClientAce.OpcDaClient.ReturnCode

SubscriptionModify ( _

      ByVal serverSubscription As Integer, _

      ByVal updateRate As Integer, _

      ByRef revisedUpdateRate As Integer _
```

```
) Kepware.ClientAce.OpcDaClient.ReturnCode

SubscriptionModify ( _

        ByVal serverSubscription As Integer, _

        ByVal deadband As Single _

) Kepware.ClientAce.OpcDaClient.ReturnCode
```

```
[C#]

Kepware.ClientAce.OpcDaClient.ReturnCode SubscriptionModify (

        int serverSubscription,

        bool active,

        int updateRate,

        out int revisedUpdateRate,

        float deadband

);

Kepware.ClientAce.OpcDaClient.ReturnCode SubscriptionModify (

        int serverSubscription,

        bool active

);

Kepware.ClientAce.OpcDaClient.ReturnCode SubscriptionModify (

        int serverSubscription,

        int updateRate,

        out int revisedUpdateRate

);

Kepware.ClientAce.OpcDaClient.ReturnCode SubscriptionModify (

        int serverSubscription,

        float deadband

);
```

The SubscriptionModify method is used to modify the properties of an existing subscription created with the Subscribe method. There are three overloads available to change the active, UpdateRate and Deadband subscription properties separately.

Parameter	Functionality
serverSubscription	This parameter identifies the subscription within the API. This handle was returned by the Subscribe method when the subscription was created. The API will throw an exception if an

	invalid handle is specified.
active	This parameter is used to make the subscription as active or inactive. When the subscription is active, the server will scan the items and provide data change notifications.
updateRate	This parameter is used to specify the rate at which the server scans the subscribed items. This is a requested rate: the actual update rate will be decided by the server at the time of this call, and can vary depending on demands on the server and data source. Update rate values must be in milliseconds.
revisedUpdateRate	This out parameter returns the update rate set by the OPC server, which can be different from the requested updateRate. The revised update rate will be in milliseconds.
deadband	The deadband parameter specifies the minimum deviation needed for the server to notify the client of a change of value. The deadband is given a percent (0.0–100.0) of the range of the value. The range is given by the EU Low and EU High properties of the item. A deadband of 0.0 will result in the server notifying the client of all changes in the item's value. The API will throw an exception if an invalid deadband value is specified.

Note: The return code indicates the overall success of the call. If the code indicates an item-specific error (ITEMERROR), each of the ReturnID objects should be examined in order to determine which items could not be added to the subscription and why. The return code will also indicate if the requested update rate is not supported by the server. In the event that the function cannot satisfy the request due to invalid arguments or unexpected errors, an exception will be thrown. For more information on Return Value:Return Code, refer to **ReturnCode Enumerator**.

Examples

This example modifies the properties of an existing subscription that was created with the Subscribe method.

```
[Visual Basic]

' Declare variables

Dim serverSubscription As Integer ' Assign handle return from Subscribe

Dim active As Boolean = True

Dim updateRate As Integer = 1000

Dim revisedUpdateRate As Integer

Dim deadband As Single = 0

Try

   ' Call SubscriptionModify API method

   daServerMgt.SubscriptionModify( _

   serverSubscription, _

   active, _

   updateRate, _

   revisedUpdateRate, _

   deadband)

Catch ex As Exception

   Console.WriteLine("SubscriptionModify exception. Reason: " & _

   ex.Message)
```

```
End Try
```

```csharp
[C#]

// Declare variables

int serverSubscription = 0; // Assign handle return from Subscribe

bool active = true;

int updateRate = 1000;

int revisedUpdateRate;

float deadband = 0;

try

{

      // Call SubscriptionModify API method

    daServerMgt.SubscriptionModify(serverSubscription, active, updateRate, out
    revisedUpdateRate, deadband);

}

catch (Exception ex)

{

      Console.WriteLine("SubscriptionModify exception. Reason: {0}", ex);

}
```

SubscriptionAddItems Method

```vbnet
[Visual Basic]

SubscriptionAddItems ( _

      ByVal serverSubscription As Integer, _

      ByRef itemIdentifiers() As Kepware.ClientAce.OpcDaClient.ItemIdentifier _

) As Kepware.ClientAce.OpcDaClient.ReturnCode
```

```csharp
[C#]

Kepware.ClientAce.OpcDaClient.ReturnCode SubscriptionAddItems (

      int serverSubscription,

      ref Kepware.ClientAce.OpcDaClient.ItemIdentifier[] itemIdentifiers

);
```

The SubscriptionAddItems method is used to add items to an existing subscription created with the Subscribe method.

Parameter	Functionality
serverSubscription	This parameter identifies the subscription within the API. This handle was returned by the Subscription method when the subscription was created. The API will throw an exception if an invalid handle is specified.
itemIdentifiers	The array itemIdentifiers specifies the OPC items that should be added to the subscription.

Note 1: The return code indicates the overall success of the call. If this code indicates an item-specific error (ITEMERROR), each of the ReturnID objects should be examined to determine which items could not be added to the subscription and why. In the event that the function cannot satisfy the request due to invalid arguments or unexpected errors, an exception will be thrown. For more information on Return Value:Return Code, refer to **ReturnCode Enumerator**.

Note 2: The server will send an initial update for all items added to an active subscription.

Note 3: In order for the server to return item values with a particular data type, that particular type must be requested with the ItemIdentifier.DataType property. The ResultID will indicate if the server is able to provide the value as the requested type. If the requested type cannot be provided, the values will be sent in their canonical (default) data type.

Examples

This example adds the items **Channel_1.Device_1.Tag_3** and **Channel_1.Device_1.Tag_4** to an existing subscription, created with the Subscribe method.

```vbnet
[Visual Basic]

' Declare variables

Dim serverSubscription As Integer ' Assign handle return from Subscribe

Dim itemIdentifiers(1) As Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_3"

itemIdentifiers(0).ClientHandle = 3 ' Assign unique handle

itemIdentifiers(1) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(1).ItemName = "Channel_1.Device_1.Tag_4"

itemIdentifiers(1).ClientHandle = 4 ' Assign unique handle

Try

   ' Call SubscriptionAddItems API method

   daServerMgt.SubscriptionAddItems( _

   serverSubscription, _

   itemIdentifiers)

   ' Check item results

   Dim item As Kepware.ClientAce.OpcDaClient.ItemIdentifier

   For Each item In itemIdentifiers

      If item.ResultID.Succeeded = False Then
```

```
            Console.WriteLine("SubscriptionAddItems failed for item: " & _

                item.ItemName)

        End If

    Next

Catch ex As Exception

    Console.WriteLine("SubscriptionAddItems exception. Reason: " & _

        ex.Message)

End Try
```

```
[C#]

// Declare variables

int serverSubscription = 0; // Assign handle return from Subscribe

ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];

itemIdentifiers[0] = new ItemIdentifier();

itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_3";

itemIdentifiers[0].ClientHandle = 3; // Assign unique handle

itemIdentifiers[1] = new ItemIdentifier();

itemIdentifiers[1].ItemName = "Channel_1.Device_1.Tag_4";

itemIdentifiers[1].ClientHandle = 4; // Assign unique handle

ReturnCode returnCode;

try

{ // Call SubscriptionAddItems API method

    returnCode = daServerMgt.SubscriptionAddItems(serverSubscription, ref
    itemIdentifiers);

     // Check item results

    if (returnCode != ReturnCode.SUCCEEDED)

    {

            foreach (ItemIdentifier item in itemIdentifiers)

            {

                    if (!item.ResultID.Succeeded)

                    {
```

```
                            Console.WriteLine("SubscriptionAddItems failed for
                            item: {0}", item.ItemName);

                    }

            }

        }

}

catch (Exception ex)

{ Console.WriteLine("SubscriptionAddItems exception. Reason: {0}", ex) }
```

SubscriptionRemoveItems Method

```
[Visual Basic]

SubscriptionRemoveItems ( _

        ByVal serverSubscription As Integer, _

        ByRef itemIdentifiers() As Kepware.ClientAce.OpcDaClient.ItemIdentifier _

) As Kepware.ClientAce.OpcDaClient.ReturnCode
```

```
[C#]

Kepware.ClientAce.OpcDaClient.ReturnCode SubscriptionRemoveItems (

        int serverSubscription,

        ref Kepware.ClientAce.OpcDaClient.ItemIdentifier[] itemIdentifiers

);
```

The SubscriptionRemoveItems method removes items from an existing subscription that was created with the Subscribe method.

Parameter	Functionality
serverSubscription	This parameter identifies the subscription within the API. This handle was returned by the Subscribe method when the subscription was created. The API will throw an exception if an invalid handle is specified.
itemIdentifiers	The array itemIdentifiers specifies the OPC items that should be removed from the Subscription.

The return code indicates the overall success of the call. If the code indicates an item-specific error (ITEMERROR), each of the ReturnID objects should be examined in order to determine which items could not be removed from the subscription and why. In the event that the function cannot satisfy the request due to invalid arguments or unexpected errors, an exception will be thrown. For more information on Return Value:Return Code, refer to **ReturnCode Enumerator**.

Examples
This example removes the items **Channel_1.Device_1.Tag_1** and **Channel_1.Device_1.Tag_2** from an existing subscription that was created with the Subscribe method.

```
[Visual Basic]
```

```vb
' Declare variables

Dim serverSubscription As Integer ' Assign handle return from Subscribe

Dim itemIdentifiers(1) As Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_3"

itemIdentifiers(0).ClientHandle = 3 ' Assign unique handle

itemIdentifiers(1) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(1).ItemName = "Channel_1.Device_1.Tag_4"

itemIdentifiers(1).ClientHandle = 4 ' Assign unique handle

Try

    ' Call SubscriptionRemoveItems API method

    daServerMgt.SubscriptionRemoveItems( _

    serverSubscription, _

    itemIdentifiers)


    ' Check item results

    Dim item As Kepware.ClientAce.OpcDaClient.ItemIdentifier

    For Each item In itemIdentifiers

        If item.ResultID.Succeeded = False Then

            Console.WriteLine( _

              "SubscriptionRemoveItems failed for item: " & _

                item.ItemName)

        End If

    Next

Catch ex As Exception

    Console.WriteLine("SubscriptionRemoveItems exception. Reason: " & _

        ex.Message)

End Try
```

```csharp
[C#]

// Declare variables
```

```
int serverSubscription = 0; // Assign handle return from Subscribe

ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];

itemIdentifiers[0] = new ItemIdentifier();

itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_3";

itemIdentifiers[0].ClientHandle = 3; // Assign unique handle

itemIdentifiers[1] = new ItemIdentifier();

itemIdentifiers[1].ItemName = "Channel_1.Device_1.Tag_4";

itemIdentifiers[1].ClientHandle = 4; // Assign unique handle

ReturnCode returnCode;

try

{ // Call SubscriptionRemoveItems API method

        returnCode = daServerMgt.SubscriptionRemoveItems(serverSubscription,

            ref itemIdentifiers);

        // Check item results

        if (returnCode != ReturnCode.SUCCEEDED)

        {

                foreach (ItemIdentifier item in itemIdentifiers)

                {

                        if (!item.ResultID.Succeeded)

                        {

                                Console.WriteLine("SubscriptionRemoveItems failed for
                                item: {0}", item.ItemName);

                        }

                }

        }

}

catch (Exception ex)

{ Console.WriteLine("SubscriptionRemoveItems exception. Reason: {0}", ex); }
```

SubscriptionCancel Method

```
[Visual Basic]
```

```
SubscriptionCancel ( _

    ByVal serverSubscription As Integer _

) As Kepware.ClientAce.OpcDaClient.ReturnCode
```

[C#]

```
Kepware.ClientAce.OpcDaClient.ReturnCode SubscriptionCancel (

    int serverSubscription

);
```

The SubscriptionCancel method is used to cancel an existing subscription created with the Subscribe method.

Parameter	Functionality
serverSubscription	This parameter identifies the subscription within the API. This handle was returned by the Subscribe method when the subscription was created. The API will throw an exception if an invalid handle is specified.

Note: In the event that the function cannot satisfy the request due to invalid arguments or unexpected errors, an exception will be thrown. For more information on Return Value: Return Code, refer to **ReturnCode Enumerator**.

Examples

```
[Visual Basic]

' Declare variables

Dim serverSubscription As Integer ' Assign handle return from Subscribe

Try

    daServerMgt.SubscriptionCancel(serverSubscription)

Catch ex As Exception

    Console.WriteLine("SubscriptionCancel exception. Reason: " & _

        ex.Message)

End Try
```

```
[C#]

// Declare variables

int serverSubscription = 0; // Assign handle return from Subscribe

try

{

    // Call SubscriptionCancel API method

    daServerMgt.SubscriptionCancel(serverSubscription);

}
```

```
catch ( Exception ex)

{

        Console. WriteLine( "SubscriptionCancel exception. Reason: {0}", ex);

}
```

WriteAsync Method

```
[ Visual basic]

WriteAsync( _

        ByVal transactionHandle As Integer, _

        ByRef itemIdentifiers() As kepware. ClientAce. OpcDaClient. ItemIdentifier, _

        ByVal itemValues() As kepware. ClientAce. OpcDaClient. ItemValue _

) As Kepware. ClientAce. OpcDaClient. ReturnCode
```

```
[ C#]

Kepware. ClientAce. OpcDaClient. ReturnCode WriteAsync (

        int transactionHandle,

        ref Kepware. ClientAce. OpcDaClient. ItemIdentifier[] itemIdentifiers,

        Kepware. ClientAce. OpcDaClient. ItemValue[] itemValues

);
```

Parameter	Functionality
transactionHandle	The API will return the specified handle along with the requested values in a WriteCompleted event. Thus, a WriteCompleted event can be correlated with a particular call to WriteAsync.
itemIdentifiers	The array of itemIdentifiers is used to specify the OPC items that should be read. Possible item-specific errors will be returned in the ResultID object of the associated ItemIdentifier. The API will also set the ServerHandle property. It is recommended that ItemIdentifier objects be stored if repeated reads and writes of the same objects are intended. The API will make use of the ServerHandle values to optimize OPC calls to the server.
itemValues	The array itemValues contains the Values to be written to the OPC server.

Note 1: The return code indicates the overall success of the call. If this code indicates an item-specific error (such as, ITEMERROR or ITEMANDQUALITYERROR), each of the ReturnID objects should be examined in order to determine which items could not be read and why. In the event that the function cannot satisfy the request (due to invalid arguments or unexpected errors) an exception will be thrown. For more information on Return Value:Return Code, refer to **ReturnCode Enumerator**.

Note 2: More than one item may be written at a time with the WriteAsync method. Because single multi-item writes can be executed more efficiently than a series of single-item writes, using multi-item writes is reccomended whenever it is possible.

Examples

This example writes the value "111" to tag **Channel_1.Device_1.Tag_1**, and "222" to tag **Channel_1.Device_1. Tag_2**.

```vb
[ Visual Basic]

' Declare variables

Dim transactionHandle As Integer = 0

Dim itemIdentifiers(1) As Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_1"

itemIdentifiers(0).ClientHandle = 1 ' Assign unique handle

itemIdentifiers(1) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(1).ItemName = "Channel_1.Device_1.Tag_2"

itemIdentifiers(0).ClientHandle = 2 ' Assign unique handle

Dim itemValues(1) As Kepware.ClientAce.OpcDaClient.ItemValue

itemValues(0) = New Kepware.ClientAce.OpcDaClient.ItemValue

itemValues(0).Value = "111"

itemValues(1) = New Kepware.ClientAce.OpcDaClient.ItemValue

itemValues(1).Value = "222"

Dim returnCode As Kepware.ClientAce.OpcDaClient.ReturnCode

Try

   ' Call WriteAsync API method

    returnCode = daServerMgt.WriteAsync( transactionHandle, itemIdentifiers, _

   itemValues)

   ' Check result

   If returnCode <> _

      Kepware.ClientAce.OpcDaClient.ReturnCode.SUCCEEDED Then

      Console.WriteLine("Write request failed for one or more items")

' Examine ResultID objects for detailed information.

   End If

Catch ex As Exception

   Console.WriteLine("WriteAsync exception. Reason: " & ex.Message)

End Try
```

```csharp
[C#]

// Declare variables

int transactionHandle = 0;

ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];

itemIdentifiers[0] = new ItemIdentifier();

itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_1";

itemIdentifiers[0].ClientHandle = 1; // Assign unique handle

itemIdentifiers[1] = new ItemIdentifier();

itemIdentifiers[1].ItemName = "Channel_1.Device_1.Tag_2";

itemIdentifiers[1].ClientHandle = 2; // Assign unique handle

ItemValue[] itemValues = new ItemValue[2];

itemValues[0] = new ItemValue();

itemValues[0].Value = "111";

itemValues[1] = new ItemValue();

itemValues[1].Value = "222";

ReturnCode returnCode;

try

{ // Call WriteAsync API method

        returnCode = daServerMgt.WriteAsync(transactionHandle, ref
itemIdentifiers, itemValues);

        // Check item results

        if (returnCode != ReturnCode.SUCCEEDED)

        { Console.WriteLine("Write request failed for one or more items");

                // Examine ResultID objects for detailed information.

        }

}

catch (Exception ex)

{ Console.WriteLine("WriteAsync exception. Reason: {0}", ex); }
```

Write Method

```
[Visual Basic]

<

Write ( _

      ByRef itemIdentifiers() As Kepware.ClientAce.OpcDaClient.ItemIdentifier, _

      ByVal itemValues() As Kepware.ClientAce.OpcDaClient.ItemValue _

) As Kepware.ClientAce.OpcDaClient.ReturnCode
```

```
[C#]

Kepware.ClientAce.OpcDaClient.ReturnCode Write (

      ref Kepware.ClientAce.OpcDaClient.ItemIdentifier[] itemIdentifiers,

      Kepware.ClientAce.OpcDaClient.ItemValue[] itemValues

);
```

The Write method is used to write one or more values to the OPC server.

Parameter	Functionality
itemIdentifiers	The array of itemIdentifiers is used to specify the OPC items that should be written. Possible item-specific errors will be returned in the ResultID object of the associated ItemIdentifier. The API will also set the ServerHandle property. It is recommended that ItemIdentifier objects be stored if repeated reads and writes of the same objects are intended. The API will make use of the ServerHandle values to optimize OPC calls to the server.
itemValues	The array itemValues contains the values to be written to the OPC server.

Note 1: The return code indicates the overall success of the call. If this code indicates an item-specific error(such as, ITEMERROR), each of the ReturnID objects should be examined in order to determine which items could not be read and why. In the event that the function cannot satisfy the request (due to invalid arguments or unexpected errors) an exception will be thrown. For more information on Return Value: Return Code, refer to **ReturnCode Enumerator**.

Note 2: Because single multi-item writes can be executed more efficiently than a series of single-item writes, using multi-item writes is recommended whenever it is possible.

Examples

This example writes the value "111" to tag **Channel_1.Device_1.Tag_1**, and "222" to tag **Channel_1.Device_1. Tag_2**.

```
[Visual Basic]

' Declare variables

Dim itemIdentifiers(1) As Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_1"

itemIdentifiers(1) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
```

```vb
itemIdentifiers(1).ItemName = "Channel_1.Device_1.Tag_2"

Dim itemValues(1) As Kepware.ClientAce.OpcDaClient.ItemValue

itemValues(0) = New Kepware.ClientAce.OpcDaClient.ItemValue

itemValues(0).Value = "111"

itemValues(1) = New Kepware.ClientAce.OpcDaClient.ItemValue

itemValues(1).Value = "222"

Try

    ' Call Write API method

    daServerMgt.Write(itemIdentifiers, itemValues)

    ' Check item results

    Dim item As Kepware.ClientAce.OpcDaClient.ItemIdentifier

    For Each item In itemIdentifiers

        If item.ResultID.Succeeded = False Then

            Console.WriteLine("Write failed for item: " & item.ItemName)

        End If

    Next

Catch ex As Exception

    Console.WriteLine("Write exception. Reason: " & ex.Message)

End Try
```

```csharp
[C#]

// Declare variables

ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];

itemIdentifiers[0] = new ItemIdentifier();

itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_1";

itemIdentifiers[1] = new ItemIdentifier();

itemIdentifiers[1].ItemName = "Channel_1.Device_1.Tag_2";

ItemValue[] itemValues = new ItemValue[2];

itemValues[0] = new ItemValue();

itemValues[0].Value = "111";

itemValues[1] = new ItemValue();
```

```
itemValues[1].Value = "222";

ReturnCode returnCode;

try

{   // Call Write API method

        returnCode = daServerMgt.Write(ref itemIdentifiers, itemValues);

        // Check item results

        if (returnCode != ReturnCode.SUCCEEDED)

         {   foreach (ItemIdentifier item in itemIdentifiers)

                {

                        if (!item.ResultID.Succeeded)

                        {

                                Console.WriteLine("Write failed for item: {0}", item.
                                ItemName);

                        }

                }

        }

}

catch (Exception ex)

{   Console.WriteLine("Write exception. Reason: {0}", ex);   }
```

ReadAsync Method

```
[Visual Basic]

ReadAsync ( _

ByVal transactionHandle As Integer, _

ByVal maxAge As Integer, _

ByRef itemIdentifiers() as Kepware.ClientAce.OpcDaClient.ItemIdentifier _

) As Kepware.ClientAce.OpcDaClient.ReturnCode
```

```
[C#]

Kepware.ClientAce.OpcDaClient.ReturnCode ReadAsync (

int transactionHandle,

int maxAge,
```

```
ref Kepware.ClientAce.OpcDaClient.ItemIdentifier[] itemIdentifiers

);
```

Items of an OPC Server can be read asynchronously using the ReadAsync method. The read values are returned in the ReadCompleted event. It is strongly recommended that a Subscription be used if the items are read cyclically (and the changed data be received in the DataChanged event).

Parameter	Functionality
maxAge	Specifies whether or not the server should return a value from cache or from the device for the specified items. If the freshness of the items cached value is within the maxAge, the cache value will be returned. Otherwise, the server will obtain the data from device. The value of maxAge must be in milliseconds. Supported for OPC DA 3.0 servers only **Note:** If maxAge is set to 0, the server will always obtain the data from device.
itemIdentifiers	The array of itemIdentifiers is used to specify the OPC items that should be read. Possible item-specific errors will be returned in the ResultID object of the associated ItemIdentifier. The API will also set the ServerHandle property. It is recommended that ItemIdentifier objects be stored if repeated reads and writes of the same objects are intended. The API will make use of the ServerHandle values to optimize OPC calls to the server.
transactionHandle	The API will return the specified handle along with the requested values in a ReadCompleted event. Thus, a ReadCompleted event may be correlated with a particular call to ReadAsync.

Note 1: The return code indicates the overall success of the call. If this code indicates an item-specific error (such as, ITEMERROR, QUALITYNOTGOOD or ITEMANDQUALITYERROR) each of the ReturnID objects should be examined in order to determine which items could not be read and why. In the event that the function cannot satisfy the request (due to invalid arguments or unexpected errors), an exception will be thrown. For more information on Return Value: ReturnCode, refer to **ReturnCode Enumerator**.

Note 2: The ReadAsynch method allows more than one item to be read at a time. Because single multi-item writes can be executed more efficiently than a series of single-item writes, using multi-item writes is reccomended whenever it is possible.

Note 3: If a particular data type is desired, specify **ItemIdentifier.DataType**. Because it is a requested type, it may not be honored. The ResultID of the item will indicate if the server was not able to read the item due to an unsupported data type.

Examples

This example reads two items: **Channel_1.Device_1.Tag_1** and **Channel_1.Device_1.Tag_2**.

```
[Visual Basic]

' Declare variables

Dim transactionHandle As Integer = 0

Dim maxAge As Integer = 0

Dim itemIdentifiers(1) As Kepware.ClientAce.OpcDaClient.ItemIdentifier
```

```vbnet
itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_1"

itemIdentifiers(0).ClientHandle = 1 ' Assign unique handle

itemIdentifiers(1) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(1).ItemName = "Channel_1.Device_1.Tag_2"

itemIdentifiers(1).ClientHandle = 2 ' Assign unique handle

Dim returnCode As Kepware.ClientAce.OpcDaClient.ReturnCode

Try

    ' Call ReadAsync API method

    returnCode = daServerMgt.ReadAsync( _

    transactionHandle, _

    maxAge, _

    itemIdentifiers)

    ' Check result

    If returnCode <> _

        Kepware.ClientAce.OpcDaClient.ReturnCode.SUCCEEDED Then

        Console.WriteLine("ReadAsync failed for one or more items")

        ' Examine ResultID objects for detailed information.

    End If

Catch ex As Exception

    Console.WriteLine("ReadAsync exception. Reason: " & ex.Message)

End Try
```

```csharp
[C#]

// Declare variables

int transactionHandle = 0;

int maxAge = 0;

ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];

itemIdentifiers[0] = new ItemIdentifier();

itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_1";

itemIdentifiers[0].ClientHandle = 1; // Assign unique handle
```

```
itemIdentifiers[1] = new ItemIdentifier();

itemIdentifiers[1].ItemName = "Channel_1.Device_1.Tag_2";

itemIdentifiers[1].ClientHandle = 2; // Assign unique handle

ReturnCode returnCode;

try

{

     // Call ReadAsync API method

      returnCode = daServerMgt.ReadAsync(transactionHandle, maxAge, ref
temIdentifiers);

     // Check result

     if (returnCode != ReturnCode.SUCCEEDED)

     {

             Console.WriteLine("ReadAsync failed for one or more items");

             // Examine ResultID objects for detailed information.

     }

}

catch (Exception ex)

{

Console.WriteLine("ReadAsync exception.  Reason: {0}", ex);

}
```

Read Method

```
[Visual Basic]

Read ( _

     ByVal maxAge As Integer, _

     ByRef itemIdentifiers() As Kepware.ClientAce.OpcDaClient.ItemIdentifier, _

     ByRef itemValues () As Kepware.ClientAce.OpcDaClient.ItemValue _

) As Kepware.ClientAce.OpcDaClient.ReturnCode
```

```
[C#]

Kepware.ClientAce.OpcDaClient.ReturnCode Read (

     int maxAge,
```

```
        ref Kepware.ClientAce.OpcDaClient.ItemIdentifier[] itemIdentifiers,

        out Kepware.ClientAce.OpcDaClient.ItemValue[] itemValues

);
```

The Read method is used to read one or more values from the OPC server. It is strongly recommended that a Subscription be used if the items are read cyclically (and the changed data be received in the DataChanged event).

Parameter	Functionality
maxAge	Specifies whether or not the server should return a value from cache or from the device for the specified items. If the freshness of the items cached value is within the maxAge, the cache value will be returned. Otherwise, the server will obtain the data from device. The value of maxAge must be in milliseconds. Supported for OPC DA 3.0 servers only. **Note:** If maxAge is set to 0, the server will always obtain the data from device.
itemIdentifiers	The array of itemIdentifiers is used to specify the OPC items that should be read. Possible item-specific errors will be returned in the ResultID object of the associated ItemIdentifier. The API will also set the ServerHandle property. It is recommended that ItemIdentifier objects be stored if repeated reads and writes of the same items are intended. The API will make use of the ServerHandle values to optimize OPC calls to the server.
itemValues	The array itemValues contains Value, Quality and Timestamp for each item.

Note 1: The return code indicates the overall success of the call. If this code indicates an item-specific error (such as, ITEMERROR, QUALITYNOTGOOD or ITEMANDQUALITYERROR) each of the ReturnID objects should be examined in order to determine which items could not be read and why. In the event that the function cannot satisfy the request (due to invalid arguments or unexpected errors), an exception will be thrown. For more information on Return Value: ReturnCode, refer to **ReturnCode Enumerator**.

Note 2: The Read method allows more than one item to be read at a time. Because single multi-item writes can be executed more efficiently than a series of single-item writes, using multi-item writes is reccomended whenever it is possible.

Note 3: If a particular data type is desired, specify **ItemIdentifier.DataType**. Because it is a requested type, it may not be honored. The ResultID of the item will indicate if the server was not able to read the item due to an unsupported data type.

Example

This example reads two items: **Channel_1.Device_1.Tag_1** and **Channel_1.Device_1.Tag_2**.

Visual Basic Example

```
' Declare variables

Dim maxAge As Integer = 0

Dim itemIdentifiers(1) As Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(0).ItemName = "Channel_1.Device_1.Tag_1"
```

```vb
itemIdentifiers(1) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier

itemIdentifiers(1).ItemName = "Channel_1.Device_1.Tag_2"

Dim itemValues(1) As Kepware.ClientAce.OpcDaClient.ItemValue

Try

    ' Call Read API method

    daServerMgt.Read( _

    maxAge, _

    itemIdentifiers, _

    itemValues)

    ' Handle results

    Dim item As Integer

    For item = 0 To 1

        If itemIdentifiers(item).ResultID.Succeeded = True Then

            Console.WriteLine( _

                "Value: " & itemValues(item).Value & _

                " Quality: " & itemValues(item).Quality.Name & _

                " Timestamp: " & itemValues(item).TimeStamp)

        Else

            Console.WriteLine("Read failed for item: " & _

                itemIdentifiers(item).ItemName)

        End If

    Next

Catch ex As Exception

    Console.WriteLine("Read exception. Reason: " & ex.Message)

End Try
```

C# Example

```csharp
// Declare variables

int maxAge = 0;

ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];

itemIdentifiers[0] = new ItemIdentifier();
```

```
itemIdentifiers[0].ItemName = "Channel_1.Device_1.Tag_1";

itemIdentifiers[1] = new ItemIdentifier();

itemIdentifiers[1].ItemName = "Channel_1.Device_1.Tag_2";

ItemValue[] itemValues = null;

try

{ // Call Read API method

        daServerMgt.Read(maxAge, ref itemIdentifiers, out itemValues);

        // Handle results

        for (int item = 0; item < 2; item++)

        {

                if (itemIdentifiers[item].ResultID.Succeeded)

                {

                        Console.WriteLine("Value: {0} Quality: {1} Timestamp {2}",

                                            itemValues[item].Value,

                                            itemValues[item].Quality.Name,

                                            itemValues[item].TimeStamp);

                }

                else

                {

                        Console.WriteLine("Read failed for item: {}",

                                            itemIdentifiers[item].ItemName);

                }

        }

}

catch (Exception ex)

{ Console.WriteLine("Read exception. Reason: {0}", ex); }
```

DataChanged Event

```
[Visual Basic]

DataChanged ( _
```

```
        ByVal clientSubscription As Integer, _

        ByVal allQualitiesGood As Boolean, _

        ByVal noErrors As Boolean, _

        ByVal itemValues() As Kepware.ClientAce.OpcDaClient.ItemValueCallback _

) Handles daServerMgt.DataChanged
```

```
[ C#]

Void DataChanged (

        int clientSubscription,

        bool allQualitiesGood,

        bool noErrors,

        Kepware.ClientAce.OpcDaClient.ItemValueCallback[] itemValues

);
```

Note: A DataChanged event will occur when the value or quality of one or more items in a subscription change. Implement a DataChanged event handler to receive the new item values.

Parameter	Functionality
clientSubscription	This is the handle given to the subscription when created with the Subscribe method.
allQualitiesGood	This flag will be set True if all values included in the data changed notification have good quality.
noErrors	This flag will be set True if there are no item errors, as indicated by the ResultID, in the values included in the data changed notification. If this flag is False, all ItemValue. ResultID objects should be examined to determine which items are in error and why.
itemValues	This array contains the value, quality, and timestamp that have changed. The ItemValue elements also contain ResultID objects that are used to indicate possible item-specific errors.

To add a DataChanged event handler in the Visual Basic application:

1. Declare a DaServerMgt object **WithEvents**.

2. Dim **WithEvents daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt**.

3. Allow the **Application Wizard** to generate the event handler template by selecting the **daServerMgt object** and the **DataChanged event**.

4. Implement the event handler as desired.

Note: For more information, refer to Example Code below.

To add a DataChanged event handler in the C# application:

1. Register the event with **DaServerMgt object**. **daServerMgt.DataChanged += new DAServerMgt. DataChangedEventHandler(DataChanged)**.

2. Implement the event handler function as desired.

Note: For more information, refer to the Example Code below.

Examples

```vbnet
[Visual Basic]

Try

    Dim itemValue As Kepware.ClientAce.OpcDaClient.ItemValueCallback

    For Each itemValue In itemValues

        If itemValue.ResultID.Succeeded = True Then

            Console.WriteLine( _

                "Item: " & itemValue.ClientHandle & _

                "Value: " & itemValue.Value & _

                "Quality: " & itemValue.Quality.Name & _

                "Timestamp: " & itemValue.TimeStamp)


        Else

            Console.WriteLine("Item error")

        End If

    Next

Catch ex As Exception

    Console.WriteLine("DataChanged exception. Reason: " & ex.Message)

End Try
```

```csharp
[C#]

private void DataChanged (int clientSubscription, bool allQualitiesGood, bool
noErrors, ItemValueCallback[] itemValues)

{

try

    {

            foreach (ItemValueCallback itemValue in itemValues)

            {

                if (itemValue.ResultID.Succeeded)

                {

                    Console.WriteLine(

                        "Item: {0}

                        Value: {1},
```

```
                              Quality: {2},

                              Timestamp: {3}",

                        itemValue.ClientHandle,

                        itemValue.Value,

                        itemValue.Quality.Name,

                        itemValue.TimeStamp);

                }

                else

                {

                        Console.WriteLine("Item error");

                }

        }

    }

    catch (Exception ex)

    {

            Console.WriteLine("DataChanged exception. Reason: {0}", ex);

    }

}
```

WriteCompleted Event

```
[Visual Basic]

WriteCompleted ( _

    ByVal transaction As Integer, _

    ByVal noErrors As Boolean, _

    ByVal itemResults() As Kepware.ClientAce.OpcDaClient.ItemResultCallback _

) Handles daServerMgt.WriteCompleted
```

```
[C#]

void WriteCompleted (

    int transactionHandle,

    bool noErrors,
```

```
        Kepware.ClientAce.OpcDaClient.ItemResultCallback[] itemResults

);
```

Note: A WriteCompleted event will occur when the API has completed an asynchronous write request.

To add a WriteCompleted event handler in the Visual Basic application:
1. Declare a DaServerMgt object **WithEvents**.

2. Dim **WithEvents daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt.**

3. Allow the **Application Wizard** to generate the event handler template by selecting the **daServerMgt object** and the **WriteCompleted event**.

4. Implement the event handler as desired.

Note: For more information, refer to Example Code below.

To add a WriteCompleted event handler in your C# application:
1. Register the event with **DaServerMgt object**. **daServerMgt.WriteCompleted += new DAServerMgt. WriteCompletedEventHandler(WriteCompleted).**

2. Implement the event handler function as desired.

Note: For more information, refer to Example Code below.

Parameter	Functionality
transaction	The handle for the read transaction passed to WriteAsync.
noErrors	This flag will be set True if there are no item errors, as indicated by the ResultID, in the items included in the write completed notification. If this flag is False, you should examine all ItemResultCallback. ResultID objects to determine which items are in error and why.
itemResults	This array contains the ClientHandle value and ResultID object for every written item.

Examples

```
[Visual Basic]

Try

    Dim result As Kepware.ClientAce.OpcDaClient.ItemResultCallback

    For Each result In itemResults

        If result.ResultID.Succeeded = False Then

            Console.WriteLine("Write failed for item: " & _

        result.ClientHandle)

        End If

    Next

Catch ex As Exception

    Console.WriteLine("WriteCompleted exception.  Reason: " & ex.Message)

End Try


[C#]
```

```csharp
private void WriteCompleted (int transactionHandle, bool noErrors,
ItemResultCallback[] itemResults)
{
      try
      {
            foreach (ItemResultCallback result in itemResults)
            {
                  if (!result.ResultID.Succeeded)
                  {
                        Console.WriteLine("Write failed for item: {0}",
                              result.ClientHandle);
                  }
            }
      }
      catch (Exception ex)
      {
            Console.WriteLine("WriteCompleted exception. Reason: {0}", ex);
      }
}
```

ReadCompleted Event

```vbnet
[Visual Basic]
ReadCompleted ( _
      ByVal transactionHandle As Integer, _
      ByVal allQualitiesGood As Boolean, _
      ByVal noErrors As Boolean, _
      ByVal itemValues() As Kepware.ClientAce.OpcDaClient.ItemValueCallback _
) Handles daServerMgt.ReadCompleted
```

```csharp
[C#]
void ReadCompleted (
```

```
        int transactionHandle,

        bool allQualitiesGood,

        bool noErrors,

        Kepware.ClientAce.OpcDaClient.ItemValueCallback[] itemValues

);
```

Note: A ReadCompleted event will occur when the API has completed an asynchronous read request.

To add a ReadCompleted event handler in the Visual Basic application:
1. Declare a DaServerMgt object **WithEvents**.

2. Dim **WithEvents daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt.**

3. Allow the **Application Wizard** to generate the event handler template by selecting the **daServerMgt object** and the **ReadCompleted** event.

4. Implement the event handler as desired.

Note: For more information, refer to Example Code below.

To add a ReadCompleted event handler in the C# application:
1. Register the event with **DaServerMgt object. daServerMgt.ReadCompleted += new DAServerMgt. ReadCompletedEventHandler(ReadCompleted)**.

2. Implement the event handler function as desired.

Note: For more information, refer to the Example Code below.

Parameter	Functionality
transactionHandle	The handle for the read transaction passed to ReadAsync.
allQualitiesGood	This flag will be set True if all values included in the read completed notification have good quality.
noErrors	This flag will be set True if there are no item errors, as indicated by the ResultID, in the values included in the read completed notification. If this flag is False, you should examine all ItemValue.ResultID objects to determine which items are in error and why.
itemValues	This array contains the value, quality, and timestamp of the items specified in the ReadASync request. The ItemValue elements also contain ResultID objects that are used to indicate possible item-specific errors.

Example:

```
[Visual Basic]

Try

    Dim itemValue As Kepware.ClientAce.OpcDaClient.ItemValueCallback

    For Each itemValue In itemValues

        If itemValue.ResultID.Succeeded = True Then        Console.WriteLine( _

            "Item: " & itemValue.ClientHandle & _

            "Value: " & itemValue.Value & _

            "Quality: " & itemValue.Quality.Name & _
```

```
                    "Timestamp: " & itemValue.TimeStamp)

        Else

            Console.WriteLine("Item error")

        End If

    Next

Catch ex As Exception

    Console.WriteLine("ReadCompleted exception. Reason: " & ex.Message)

End Try
```

```
[C#]

private void ReadCompleted (int transactionHandle, bool allQualitiesGood, bool
noErrors, ItemValueCallback[] itemValues)

{

        try

        {

                foreach (ItemValueCallback itemValue in itemValues)

                {

                        if (itemValue.ResultID.Succeeded)

                        {

                                Console.WriteLine(

                                        "Item: {0}

                                        Value: {1},

                                        Quality: {2},

                                        Timestamp: {3}",

                                    itemValue.ClientHandle,

                                    itemValue.Value,

                                    itemValue.Quality.Name,

                                    itemValue.TimeStamp);

                        }

                        else

                        {
```

```
                               Console.WriteLine("Item error");

                   }

           }

       }

       catch (Exception ex)

       {

               Console.WriteLine("ReadCompleted exception. Reason: {0}", ex);

       }

}
```

ServerStateChanged Event

```
[Visual Basic]

ServerStateChanged ( _

     ByVal clientHandle As Integer, _

     ByVal state As Kepware.ClientAce.OpcDaClient.ServerState _

) Handles daServerMgt.ServerStateChanged
```

```
[C#]

void ServerStateChanged (

     int clientHandle

     Kepware.ClientAce.OpcDaClient.ServerState state

);
```

A ServerStateChanged event will occur when the API has detected that the connection state with a server has changed. To moniter these changes and atke appropriate action in response, implement a ServerStateChanged event handler in the client application.

To add a ServerStateChanged event handler in the Visual Basic application:

1. Declare a DaServerMgt object **WithEvents**.

2. Dim **WithEvents daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt.**

3. Allow the **Application Wizard** to generate the event handler template by selecting the **daServerMgt object** and the **ServerStateChanges** event.

4. Implement the event handler as desired.

Note: Refer to the example code below for more information.

To add a ServerStateChanged event handler in the C# application:

1. Register the event with **DaServerMgt object.daServerMgt.ServerStateChanged+= newDAServerMgt. ServerStateChangedEventHandler(ServerStateChanged);**

2. Implement the event handler function as desired.

Note: For more information, refer to the example code below.

Examples:

Parameter	Functionality
clientHandle	This is the client handle associated with the particular server connection a state change notification is for. This handle is provided by the client though the Connect method.
state	The current status of the connection (see **ServerState Enumeration**).

Kepware.ClientAce.OPCCmn Interface of OpcServerEnum Object

The Kepware.ClientAce.OPCCmn namespace provides the following functionality:
- Enumerate the OPC servers installed on a given machine.

- Determine the CLSID from an OPC server's ProgID

See Also:
Creating OpcServerEnum Object
EnumComServer Method
ClsidFromProgID Method

Creating OpcServerEnum Object

Before using the OpcServerEnum Class, an instance of the class must be created.

```
[Visual Basic]

Dim opcServerEnum As New Kepware.ClientAce.OpcCmn.OpcServerEnum
```

```
[C#]

OpcServerEnum opcServerEnum = new

Kepware.ClientAce.OpcCmn.OpcServerEnum();
```

EnumComServer Method

```
[Visual Basic]

EnumComServer ( _

      ByVal nodeName As String, _

      ByVal returnAllServers As Boolean, _

      ByVal serverCategories() As Kepware.ClientAce.OpcCmn.ServerCategory, _

      ByRef servers() As Kepware.ClientAce.OpcCmn.ServerIdentifier _

)
```

```
[C#]

void EnumComServer (
```

```
        string nodeName,

        bool returnAllServers,

        Kepware.ClientAce.OpcCmn.ServerCategory[] serverCategories,

        Kepware.ClientAce.OpcCmn.ServerIdentifier[] servers

);
```

The EnumComServer method can be used to determine what OPC servers are accessible to a ClientAce application. These servers can exist on the same computer as the client application, or on any machine accessible on the network. The results can be filtered according to OPC server category. For more information, refer to **ServerState Enumeration**.

Parameter	Functionality
nodeName	The name or the IP address of the OPC server's host machine. (e.g. localhost, PCTest, 192.168.0.120, etc.). If this parameter is left unassigned, the local host is assumed.
returnAllServers	This flag decides whether to return all OPC Servers found on that particular machine or not. If this parameter is set to true, the array serverCategories will be ignored.
serverCategories	This parameter specifies which types of OPC servers should be returned (see also **ServerState Enumeration**).

Examples

This example browses for all OPCDA servers installed on localhost.

```
[ Visual Basic]

' Declare parameters

Dim nodeName As String = "localhost"

Dim returnAllServers As Boolean = False

Dim serverCatagories(0) As Kepware.ClientAce.OpcCmn.ServerCategory

serverCatagories(0) = New Kepware.ClientAce.OpcCmn.ServerCategory

serverCatagories(0) = Kepware.ClientAce.OpcCmn.ServerCategory.OPCDA

Dim servers() As Kepware.ClientAce.OpcCmn.ServerIdentifier

Try

    ' Call EnumComServer API method

    opcEnum.EnumComServer( _

        nodeName, _

        returnAllServers, _

        serverCatagories, _

        servers)

    ' Handle results

    Dim server As Kepware.ClientAce.OpcCmn.ServerIdentifier
```

```vb
    For Each server In servers

        Dim progID As String = server.ProgID

        Dim url As String = server.Url

        Console.WriteLine("ProgID: " & progID & " url: " & url)

    Next

Catch ex As Exception

    Console.WriteLine("Handled EnumComServer exception. Reason: " _

        & ex.Message)

End Try
```

```csharp
[C#]

// Declare parameters

string nodeName = "localhost";

bool returnAllServers = false;

ServerCategory[] serverCategories = new ServerCategory[1];

serverCategories[0] = new ServerCategory();

serverCategories[0] = ServerCategory.OPCDA;

ServerIdentifier[] servers;

try

{

      // Call EnumComServer API method

      opcEnum.EnumComServer(nodeName, returnAllServers, serverCategories, out
servers);

      // Handle results

      foreach (ServerIdentifier server in servers)

      {

            string progID = server.ProgID;

            string url = server.Url;

            Console.WriteLine("ProgID: {0} url: {1}", progID, url);

      }

}
```

```
catch ( Exception ex)

{

        Console.WriteLine("EnumComServer exception. Reason: {0}", ex);

}
```

ClsidFromProgID Method

```
[Visual Basic]

ClsidFromProgId ( _

        ByVal nodeName As String, _

        ByVal progID As String, _

        ByRef clsid As String _

)
```

```
[C#]

void ClsidFromProgId (

        string nodeName,

        string progId,

        out string clsid

);
```

The ClsidFromProgID method is used to obtain the CLSID (class ID) of an OPC server from its ProgID (programID). The server's host machine must be accessible from the client.

Parameter	Functionality
nodeName	The name or the IP address of the OPC Server's host machine, such as localhost, PCTest, 192.168.0.120, etc. If this parameter is left unassigned, the local host is assumed.
progID	The ProgID of the OPC server.
clsid	The returned CLSID of the OPC server.

```
[Visual Basic]

' Declare variables

Dim nodeName As String = "localhost"

Dim progId As String = "KEPware.KEPServerEx.V4"

Dim clsid As String



Try
```

```vb
    ' Call ClsidFromProgId API method

    opcEnum.ClsidFromProgId(nodeName, progId, clsid)


    ' Handle result

    Console.WriteLine("CLSID: " & clsid)


Catch ex As Exception

    Console.WriteLine("ClsidFromProgID exception. Reason: " & _

        ex.Message)

End Try
```

```csharp
[C#]

// Declare variables

string nodeName = "localhost";

string progId = "KEPware.OPCSampleServer";

string clsid;

try

{

      // Call ClsidFromProgId API method

      opcEnum.ClsidFromProgId(nodeName, progId, out clsid);

      // Handle result

      Console.WriteLine("CLSID: {0}", clsid);

}

catch (Exception ex)

{

      Console.WriteLine("ClsidFromProgId exception. Reason: {0}", ex);

}
```

DA Junction .NET Control

Overview_of ClientAce DA_Junction
Project Setup
Data Types Description
Signing Your Application

Overview of ClientAce DA Junction

The ClientAce DA Junction is a customized .NET control that allows a VB.NET or C# programmers to easily link OPC data to WinForm controls through a simple drag and drop interface. When building advanced custom OPC client applications that require more control over OPC functionality, **ClientAce .NET API** is recommended.

Features of the ClientAce DA Junction include:

- No detailed knowledge about OPC Data Access interfaces is required.

- The component completely covers the connection handling procedure for one or multiple OPC servers; such as, connection establishment, connection monitoring, and reconnection in case of errors.

- Conversion of OPC data from different OPC Data Access interfaces into .NET data types.

- Support for .NET WinForm controls available in Visual Studio and from most 3rd party vendors.

See Also:
DA Junction Configuration Window
A Sample Project Using DA Junction with VB.NET or C#
Licensing ClientAce
Signing Your Application

Project Setup

DA Junction Configuration Window
A Sample Project Using DA Junction with VB.NET or C#
Item Update Rate
Disable Datachange while Control Has Focus

DA Junction Configuration Window

The DA Junction Configuration Window is divided into 3 main parts:

- The **OPC Items pane**

- The **Controls pane**

- The **Connections pane** includes the General and Trigger **Connection Settings**
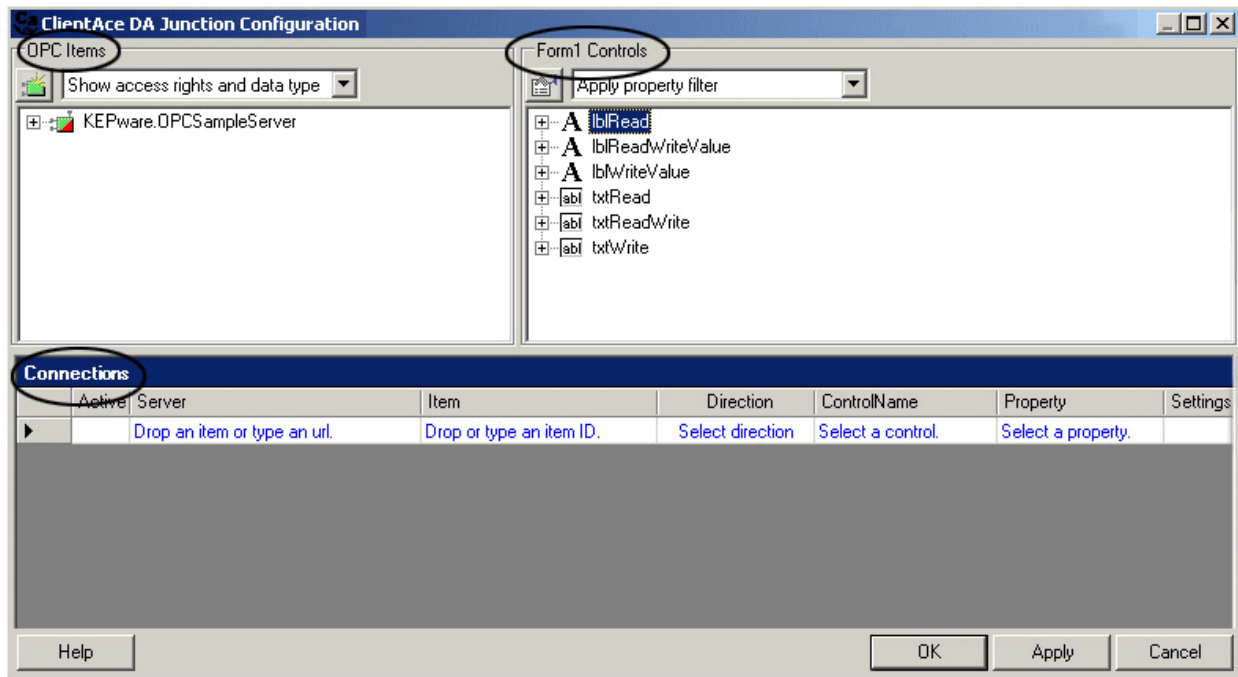
OPC Items Pane

The OPC Items pane displays items from an OPC server project.

1. Click the green browse icon.

2. Use the **Add Server dialog** to browse to the particular OPC server.
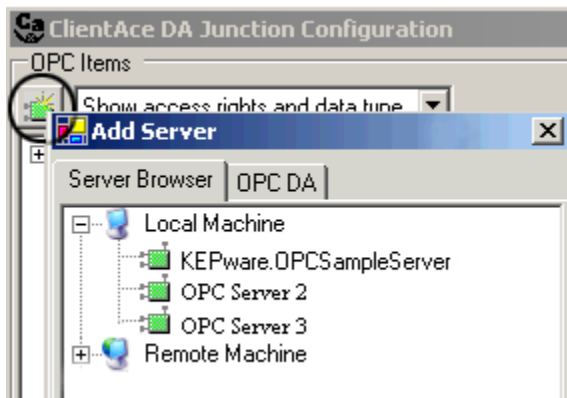
3. Click on the OPC server, then select **OK**. In the example, KEPware.OPCSampleServer is selected.
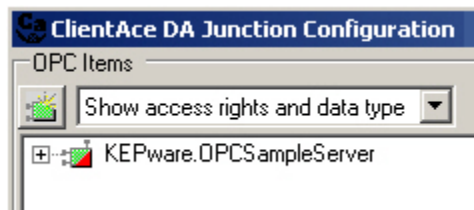
4. Use the drop-down box to choose the information displayed about the OPC server's tags; such as showing the item names (or tag names) only, showing the items' access rights and data type, and etc.
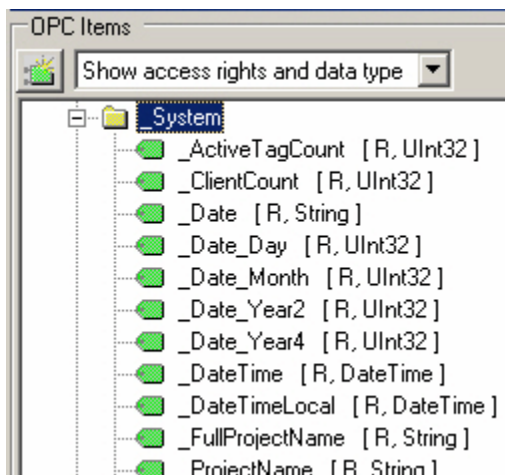
Controls Pane

The Controls pane is in the upper right area of the screen. The example shown below demonstrates the 6 controls on Form1. Use the drop-down menu to choose from the control properties being displayed.
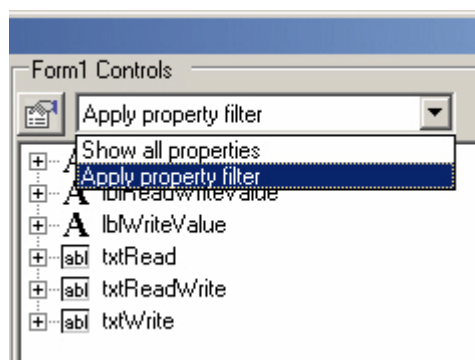
In the example shown below, **Show all properties** is selected.
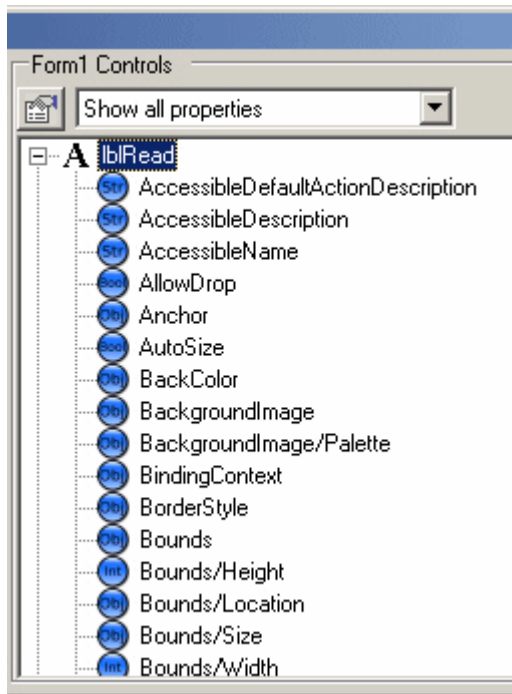
In the example shown below, the **Apply property filter**, which shows the Filter dialog, is displayed. The **Type Filter**, which includes a checklist of available data types, is found in the first tab.
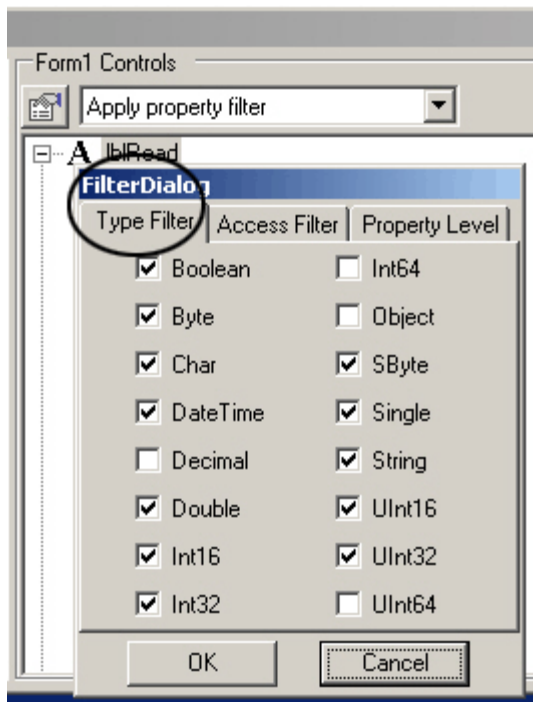
In the example shown below, the **Access Filter tab** in the Filter dialog is displayed. The **Show Read Only Properties** field is unchecked by default because data is usually written from the OPC server to the property of the user interface control. To write data from the property, **Show Read Only Properties** must be check from the OPC server.
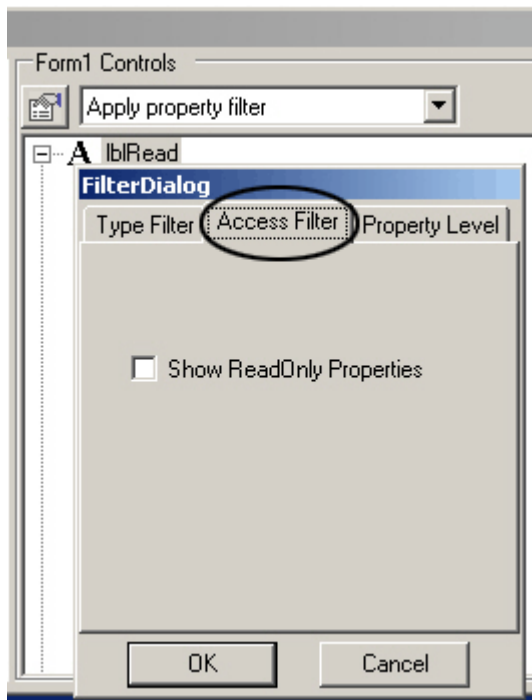
In the example shown below, the **Property Level tab** in the Filter dialog is displayed. The default level is 2. The higher the number is, the greater the level of property detail that will be shown. If the end node of a given item is at level 2, then only 2 levels will be shown for that item if the property level filter is set to 2 or higher. Likewise, if the level filter is set to 3 then only 3 levels of property detail will be shown even if a given item's end node is at level 4 or higher.
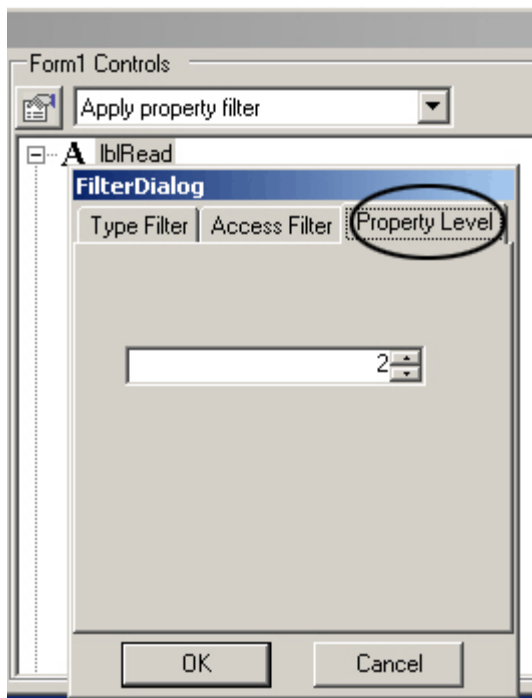
Connections Pane

The Connections pane is in the lower half of the screen. The **Connections grid** can be used to modify the tag state, server name, tag item, and data direction. It can also be used to modify or set Visual Studio controls and properties, and also to set triggers.

Direction Property

Direction is an important property when setting up the tag-control connections. The Direction property determines whether the Visual Studio control is Read Only, Write Only or Read/Write. The default is shown in **bold**.

Direction	Property	Description
Item =>Control	**Read Only**	Direction of data is from Item to Control only.
Item <= Control	Write Only	Direction of data is from Control to Item only.
Item <=> Control	Read/Write	Data flows in both directions.

Connection Settings

To access the Connection Settings for an item:
1. Click on the **Settings column**.
2. Click on the ellipses button.

Note: The **Connection Settings window** has two tabs: **General** and **Trigger**. The General tab is shown below. **See Also: Item Update Rate** and **Disable Datachange while Control Has Focus**.


The **Trigger tab** can be used to select the control, browse events and select an event that will trigger a write to the OPC tag connected to the control. For a description of the Trigger tab using a sample project, see the **Triggers** section of the Sample Project topic.
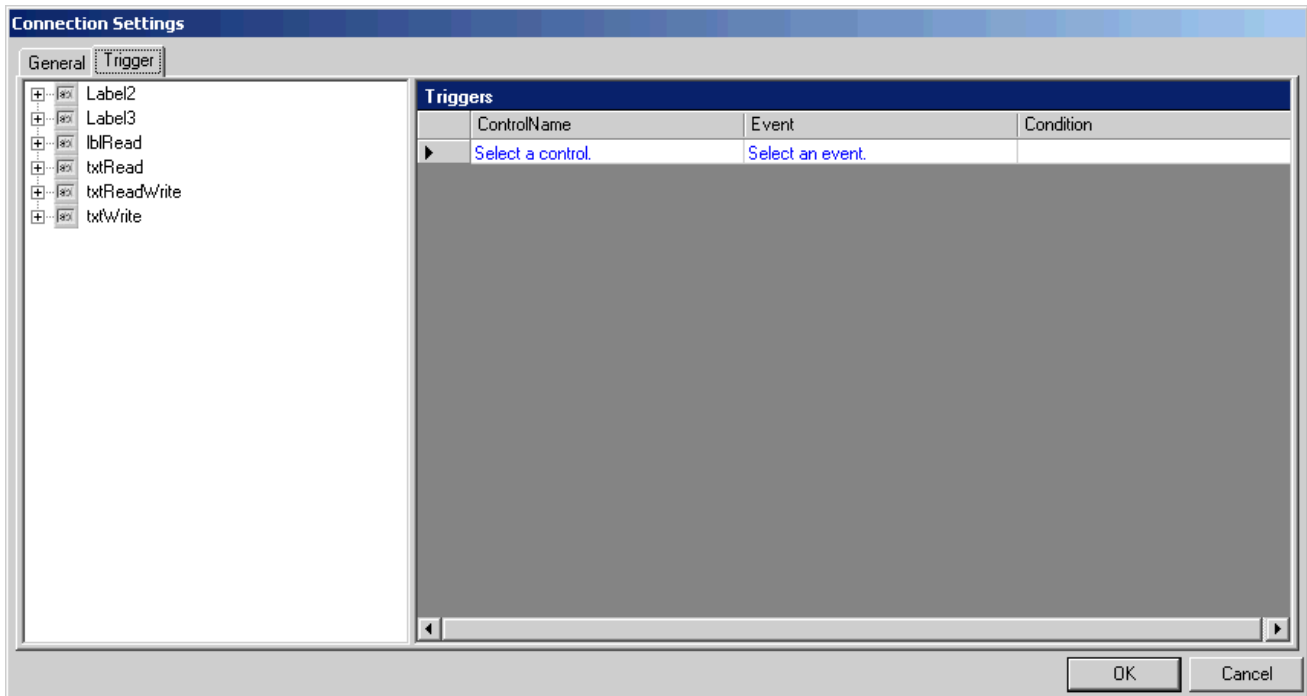
A Sample Project Using DA Junction with VB.NET or C#

Microsoft Visual Studio supports many different third-party .NET controls that can be connected to OPC tag items through the Kepware. ClientAce.DA_Junction control library. The following example demonstrates how to connect VB/C# TextBox controls to OPC tag items and then read and write to the items through the VB/C# TextBox controls.

Important: All referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, as this will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.

Step 1:
Verify that the Visual Basic Toolbox includes the ClientAceDA_Junction Control.

1. In the **Visual Basic Toolbox**, check the controls listed under the **ClientAce tab**.
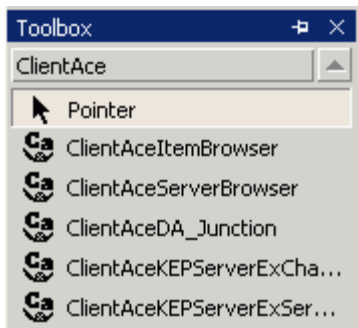
2. If the **ClientAceDA_Junction control** is missing, add it by following the procedure described in **Missing Controls.**

Step 2:
Add **VB/C# Controls** to a Windows Form.

1. Begin with a blank Form. Next, drag and drop the **ClientAceDA_Junction control** from the Toolbox to the new Form. The control label **ClientAceDA_Junction1** will be displayed in the lower left corner of the screen.
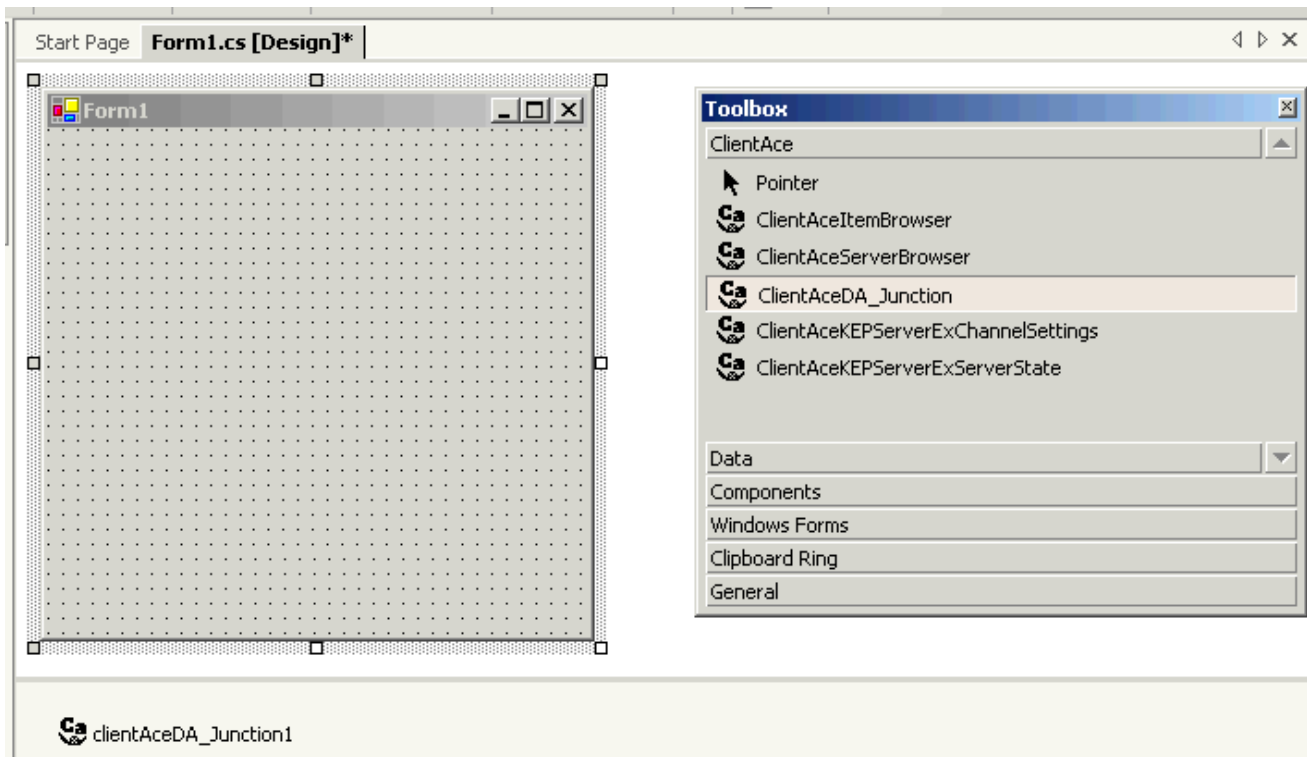
2. Drag and drop three **VB/C# Label controls** and three **TextBox controls** onto the form. The Label and TextBox controls are located under the **Windows Forms tab** in the Toolbox.
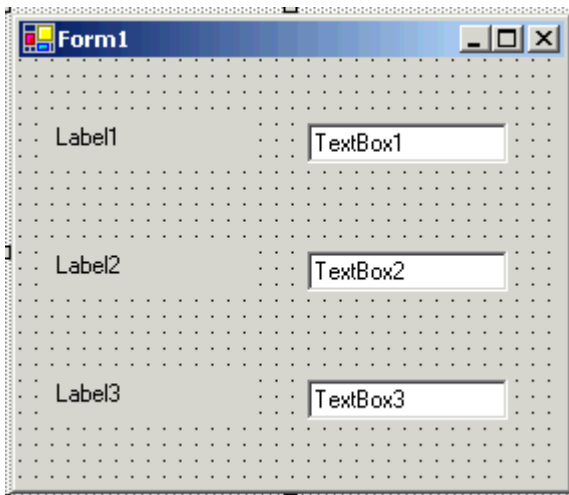
3. For this example, the name and text properties of the controls have been changed to a more descriptive name. To open **Properties**, click **View** and then select **Properties Window**. Use **ALT+ENTER** as a shortcut.

4. Click once on the Label1 Control to make sure it is selected.
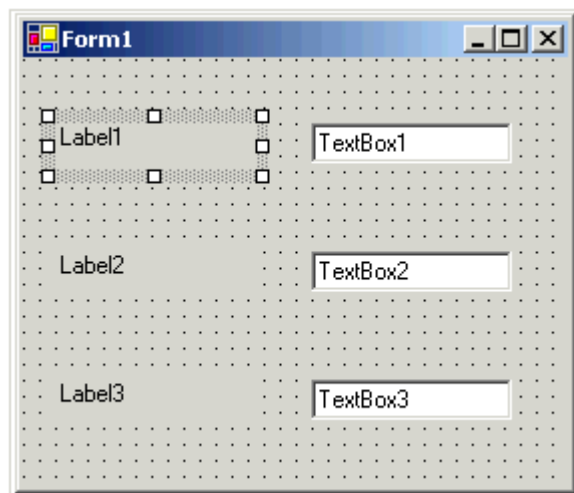
5. In the Properties window, click **Design** and then change the **Name property** of the Label1 control to **"lblRead"** (as shown below).
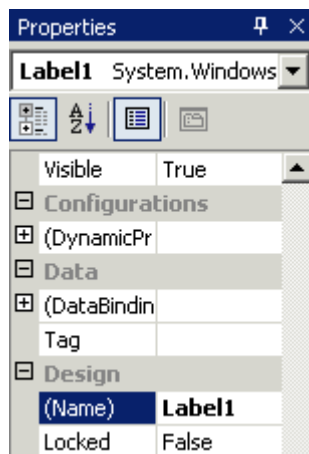
6. Under **Appearance**, change the Text property to **"ReadVal"** as shown below.
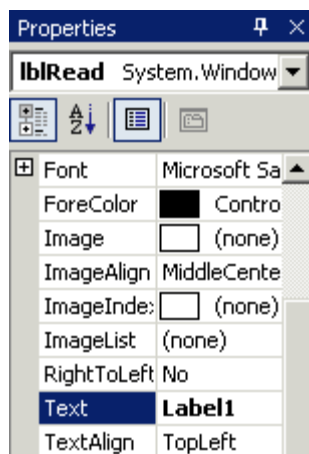
7. Repeat this procedure to change the Name and Text properties of the other 5 controls. These controls are shown displayed in the following table:

Original Default Name of Control	New (Name) Property	New Text Property
Label1	lblRead	ReadVal
Label2	lblWriteValue	WriteVal
Lable3	lblReadWriteValue	ReadWriteVal
TextBox1	txtRead	*
TextBox2	txtWrite	*
TextBox3	txtReadWrite	*

Note: The Text property for the TextBox controls should be left blank. The new Text properties will be updated automatically by the OPC tag items.

Step 3:

Call Up the ClientAce DA Junction configuration.

1. Click on the **ClientAceDA_Junction1 control** to select the **ClientAceDA_Junction1 property**.

2. In the **Properties** window, click once on the **ClientAceConfiguration property**.

3. Click on the **ellipses button** to launch the **ClientAce DA Junction Configuration window**.
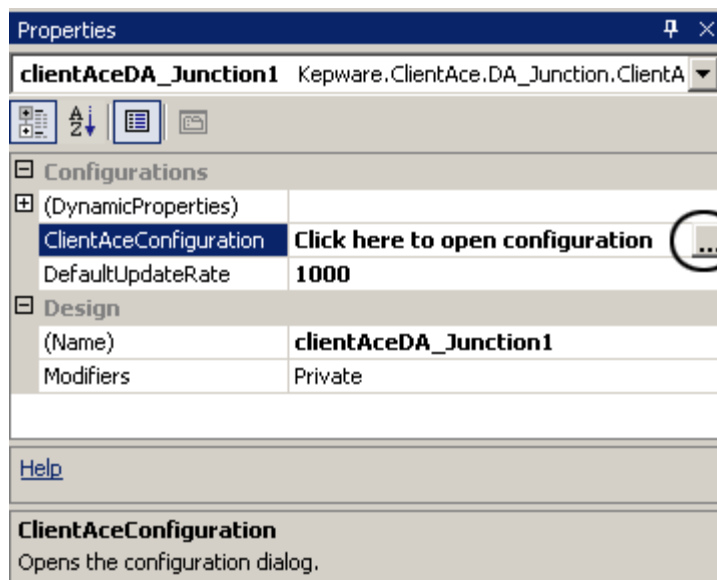
4. Use the **OPC Items pane** (on the left side of the window) to add **local** and **remote servers** and also to browse for **OPC tag items**. Use the **Control pane** (on the right side of the window) to see the **VB/C#** controls displayed. **See Also**: **DA Junction Configuration Window**).
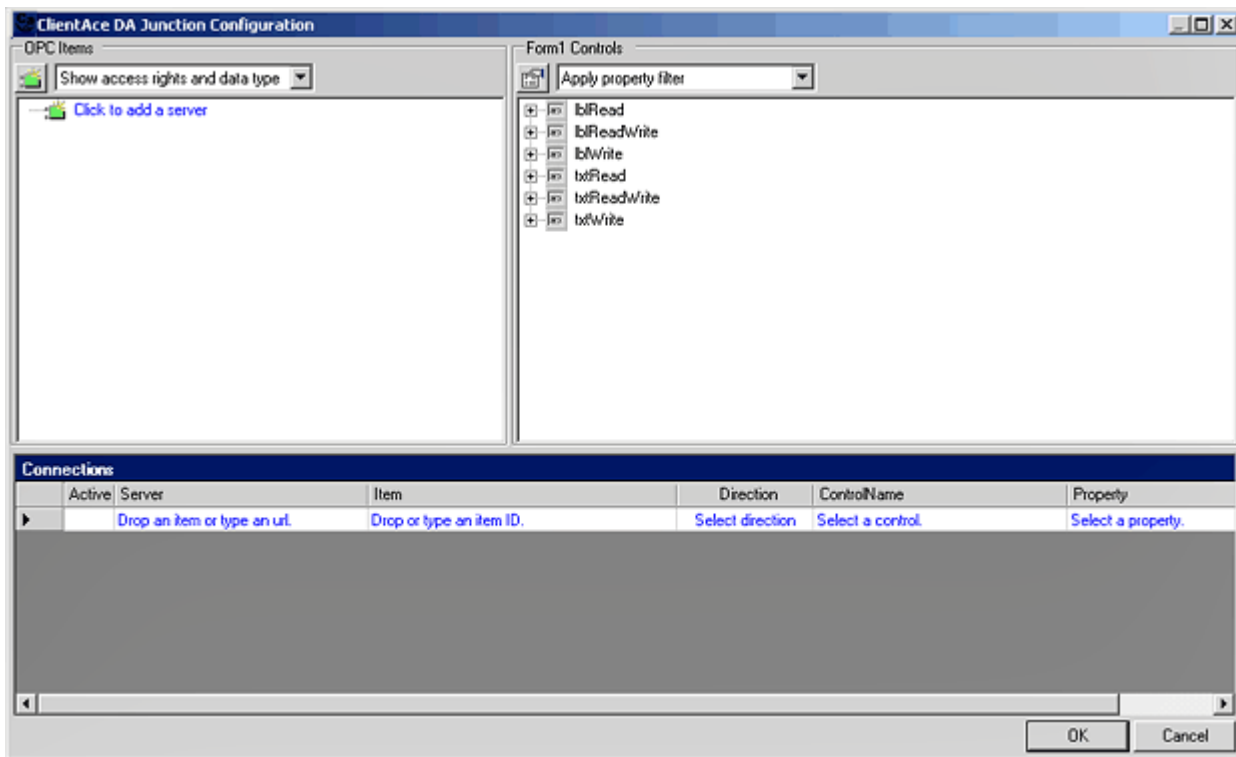
Step 4:
Connect to OPC servers and add tags.

1. Double-click on **Click to add a server link** in the left pane of the window.
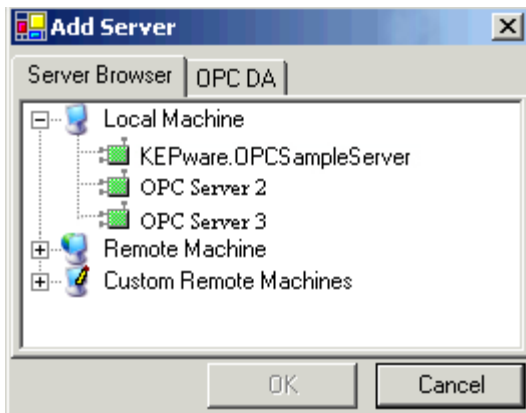
2. Select the server to connect to, either on the local computer or OPC servers on remote machines (using the nodes **Local Machine**, **Remote Machine** or **Custom Remote Machines**). In the example, the "KEPware.OPCSampleServer" OPC server is connected.

3. Browse the OPC server to get to the tags to connect with the **Visual Studio controls**.
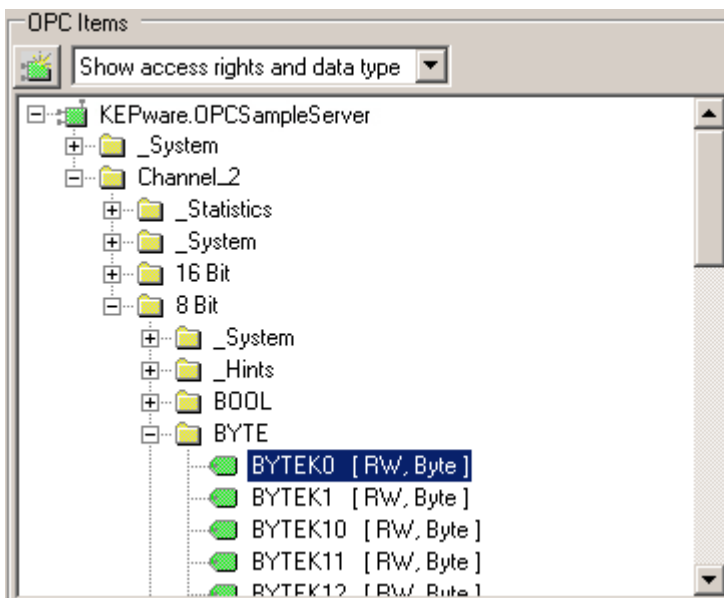
4. Drag and drop each **OPC tag item** onto the **Visual Studio control**. For example: Drag the BYTEK0 tag to the txtRead and txtWrite controls, and BYTEK1 to the txtReadWrite textbox control. Afterwards, the tag items will be listed in the **Connections grid** (at the bottom of the screen).
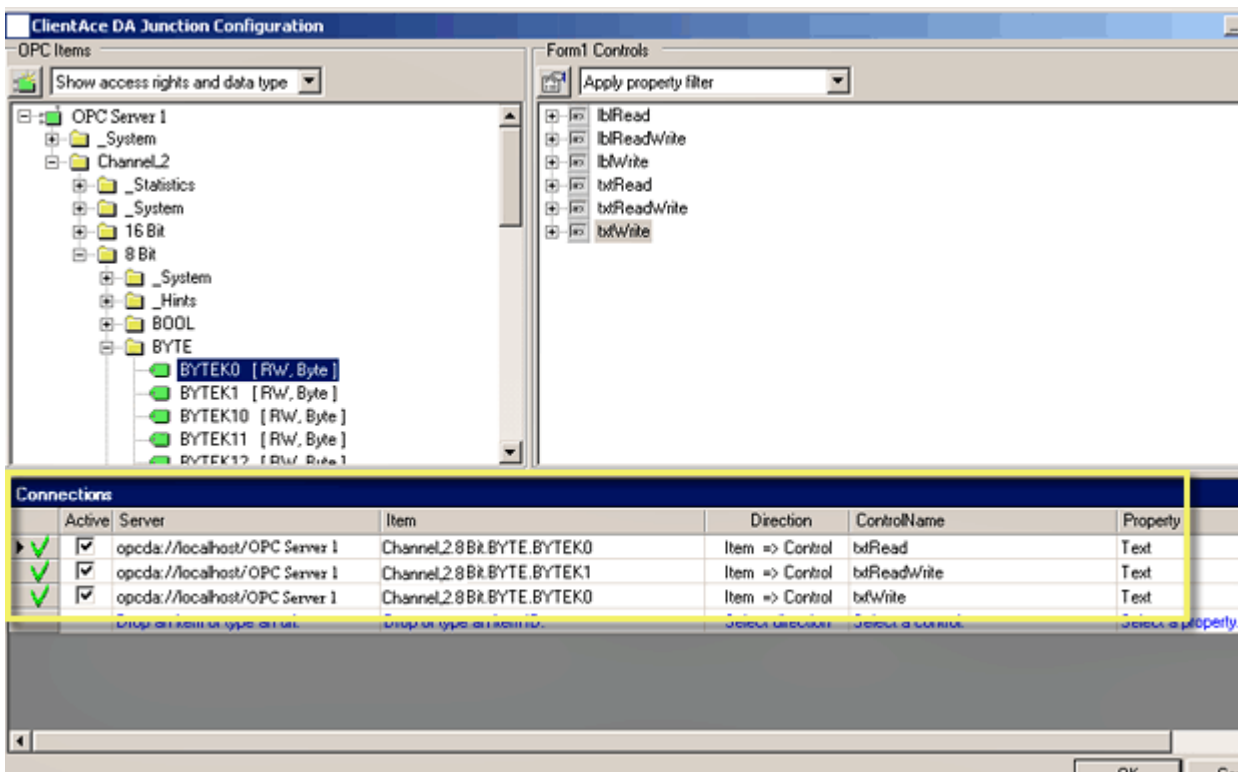
Step 5:
Modify the Connections.

Connections Grid
Use the Connections grid (at the bottom of the Configuration Window) to modify the tag state, server name, tag item, data direction, Visual Studio controls, properties, and to set triggers. (**See Also: [DA Junction Configuration Window](#)** .)

Direction Property
Direction is an important property when setting up the tag-control connections. The Direction property determines whether the Visual Studio control is Read Only, Write Only or Read/Write. The default is shown in **bold**.
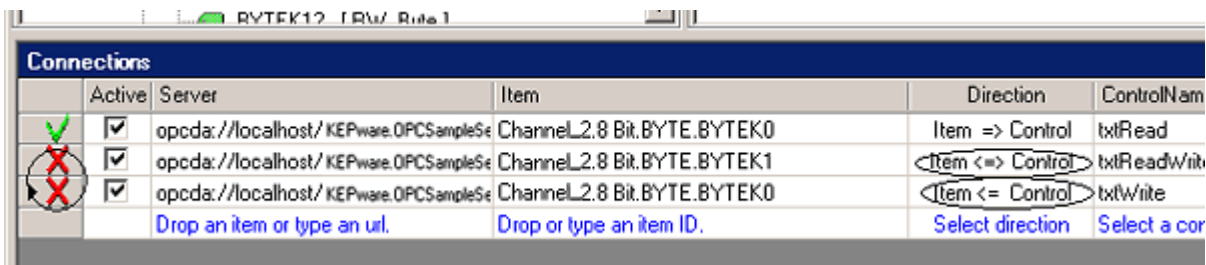
Direction	Property	Description
Item =>	**Read Only**	Direction of data is from Item to Control only.
Item <= Control	Write Only	Direction of data is from Control to Item only.
Item <=> Control	Read/Write	Data flows in both directions.

In the example, the txtRead control should be Read Only (default), the txtReadWrite control should be Read/Write, and the txtWrite control should be Write Only.

Perform the following steps:

1. Click the Direction column for the txtReadWrite control, and select Item <=> Control from the drop-down menu.

2. Click the Direction column for the txtWrite control, and select Item <= Control from the drop-down menu.

Note: When the direction is changed to Write Only (<=) or Read/Write (<=>), the item will display a red "X" in the leftmost column, as shown in the screen below. The **red X signifies an error**. This is because the control has been set to Write Only or Read/Write but the control does not yet have its write conditions specified. A property called **Triggers** can specify the conditions for the write procedures.

Triggers:
To access the Triggers property for an item:

1. Click on the **Settings** column.

2. Click the ellipses button.

3. Under the **Connection Settings** window, click the **Trigger tab**.
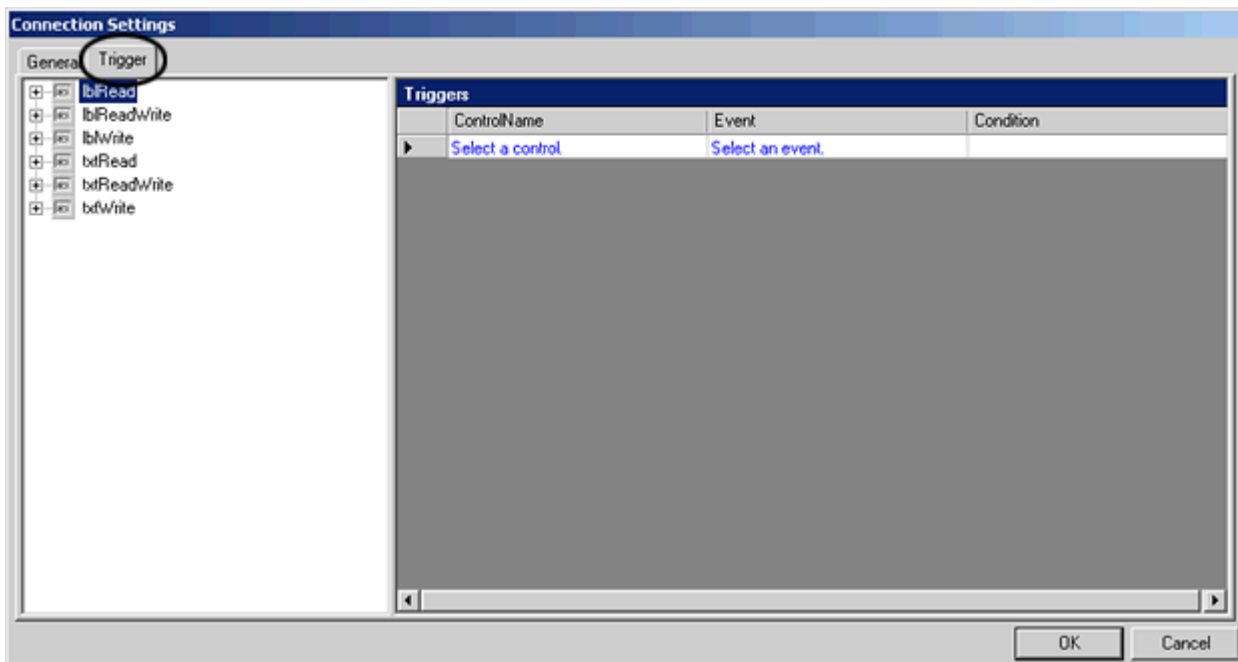
Note: The Trigger tab is used to select the control, browse events and select an event that will trigger a write to the OPC tag connected to the control. For example: The txtReadWrite and TxtWrite controls need to have their write conditions specified as follows:

- The txtReadWrite control's LostFocus event will be the event to trigger writes on the txtReadWrite Visual Studio control.

- The txtWrite control's LostFocus event will be the event to trigger writes on the txtWrite Visual Studio control.

Write Conditions

Note: Perform the following steps for txtReadWrite and txtWrite.

1. Select and expand the txtReadWrite control in the left pane of the window to see all of its properties.

2. Choose **LostFocus** from the Event drop-down list (or drag the LostFocus property and drop it in the Event column).
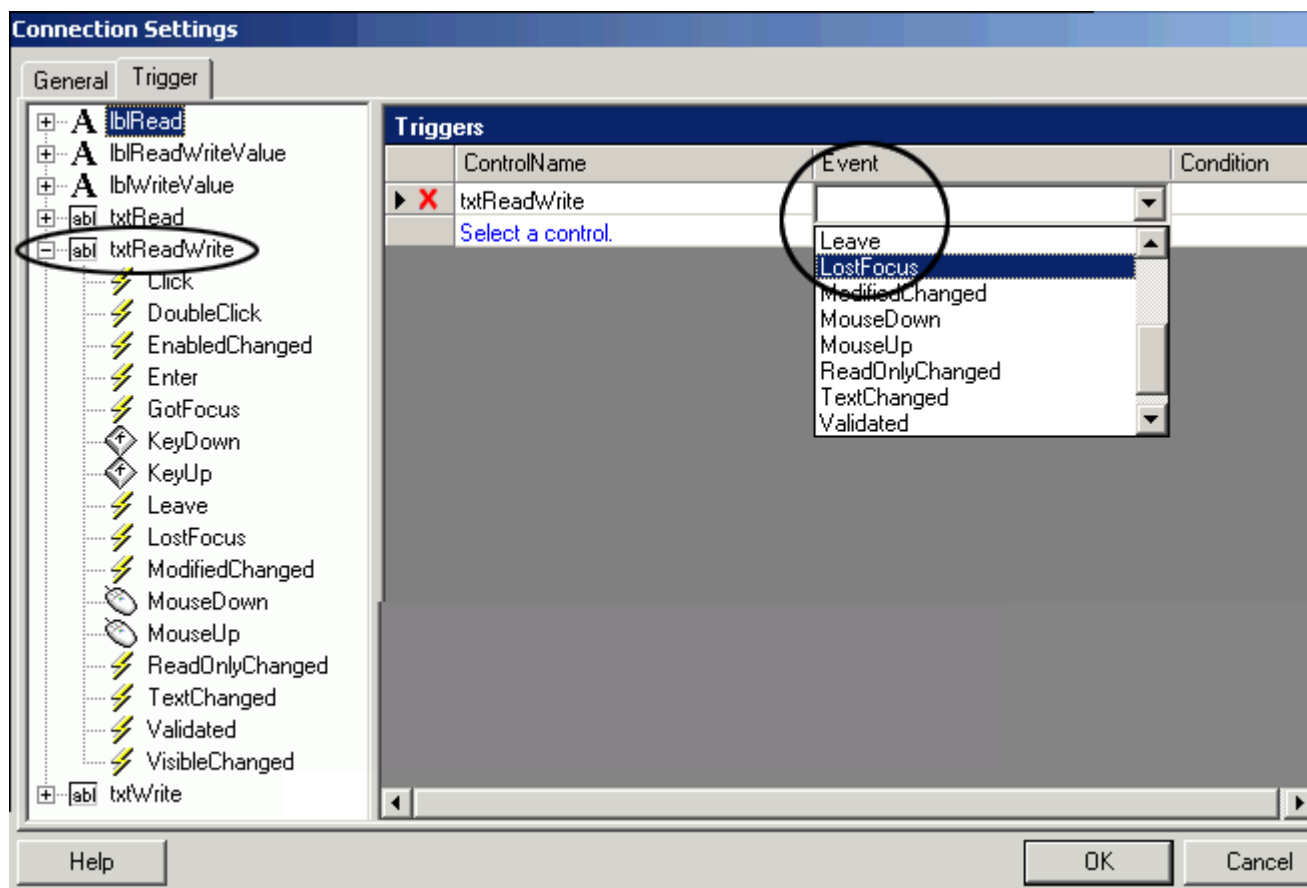
3. Click **OK**.
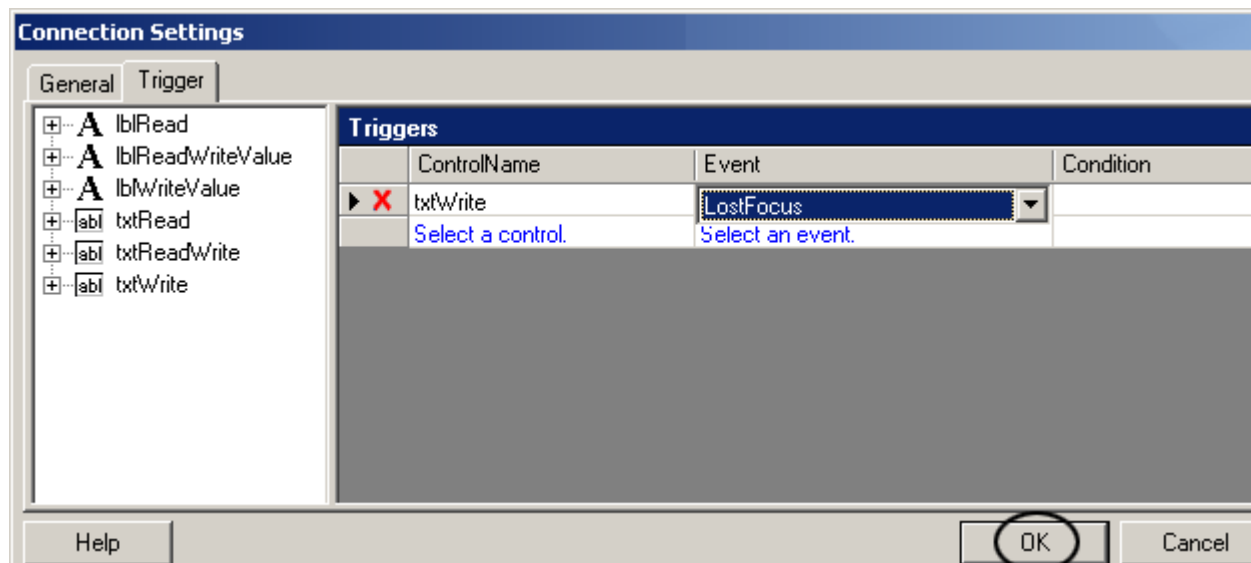
4. The **Configuration Screen** is displayed once the Connection Settings/Triggers window closes. Repeat the process for the txtWrite control.

5. In the **Connections pane**, click the **ellipses button** in the **Settings** column.

Item => Control | txtWrite | Text | ...

6. On the **Trigger** tab, select **LostFocus** as the **Event for txtWrite**.

7. Click **OK**.
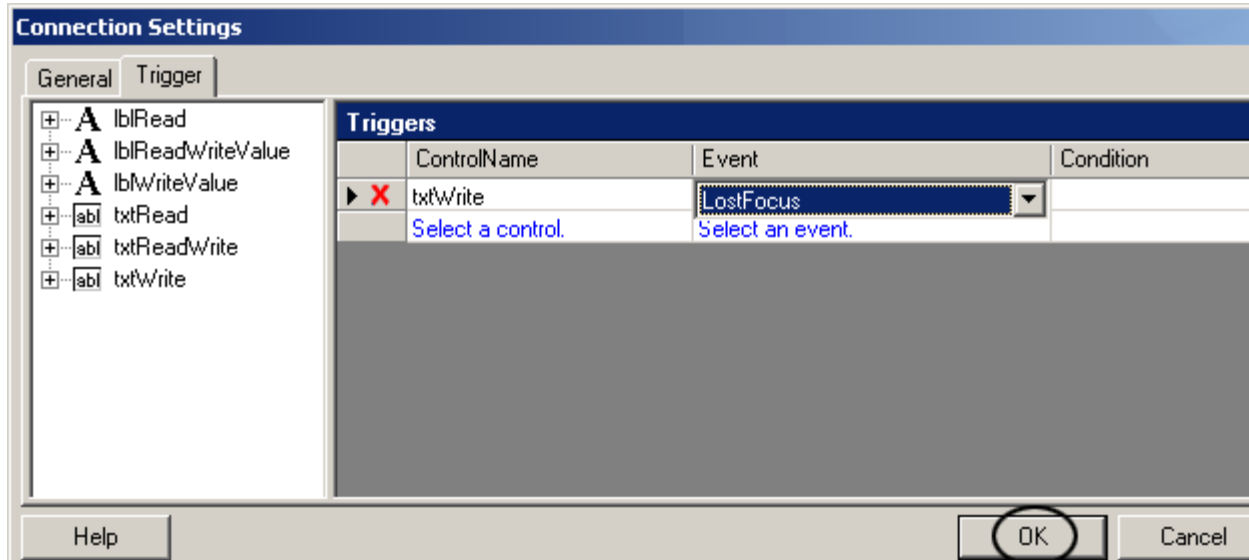
Condition Field Note: When applicable, the **Condition** field will provide a drop-down list of conditions. For example: If a control is added with KeyDown in the **Event** field, the **Condition** drop down would display a list of valid keys to choose from.
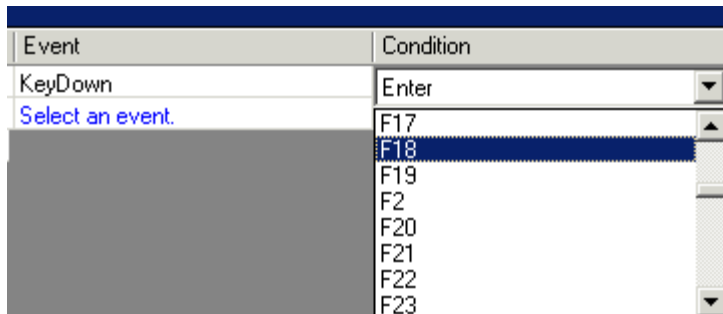
8. To finish, click **OK** at the bottom of the **Configuration screen** to save the changes made. Then, build the application and run it: it will read from and write to the OPC tags through the associated VB or C# controls.

Item Update Rate

There are two update rate settings available in ClientAce: the Global Update Rate and the Item Level Update Rate.

Default Global Update Rate for All Items

The Global Update Rate defines the default update rate for items initially added. Although the default global update rate for all items is 1000 milliseconds, it can be modified by changing the **DefaultUpdateRate property** of the DA_Junction control. An example is shown below.
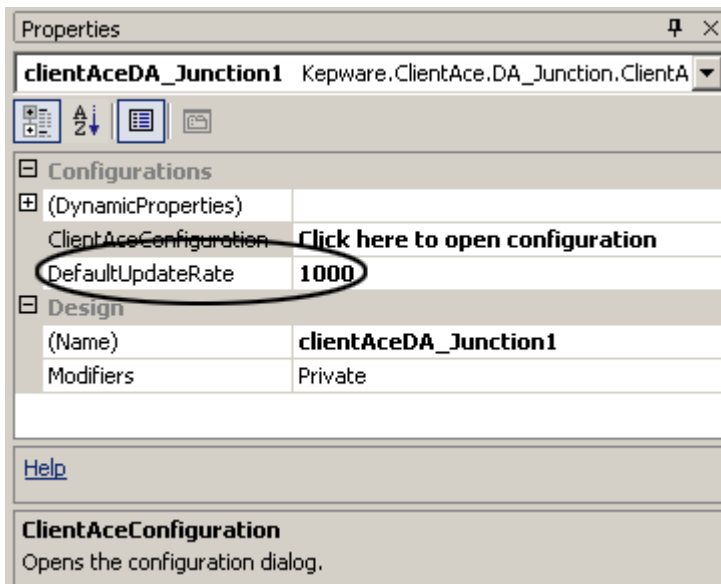
To Change the Update Rate for an Individual DA Junction Item

The update rate for an individual DA Junction item can also be changed. This change does not affect the default update rate for other controls.

1. Launch the **Configuration window** by clicking on the **ClientAceConfigurationellipses button**.
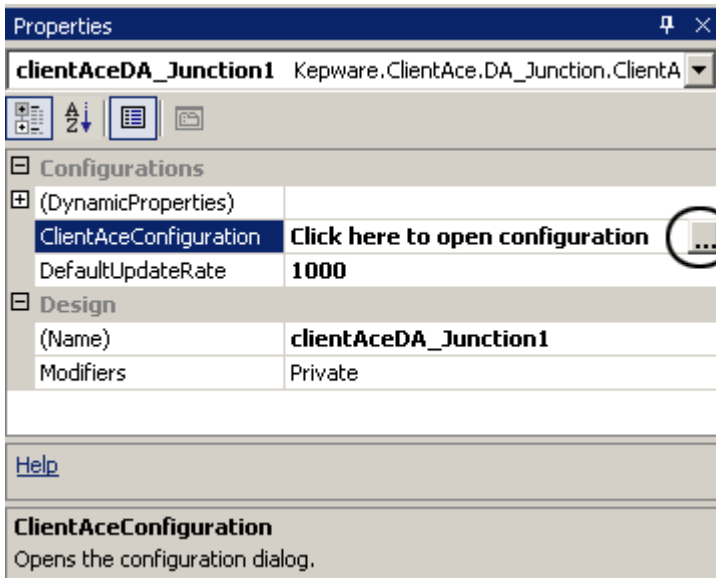
2. Click in the **Settings column** and select the ellipses next to the item whose default rate you want to change.
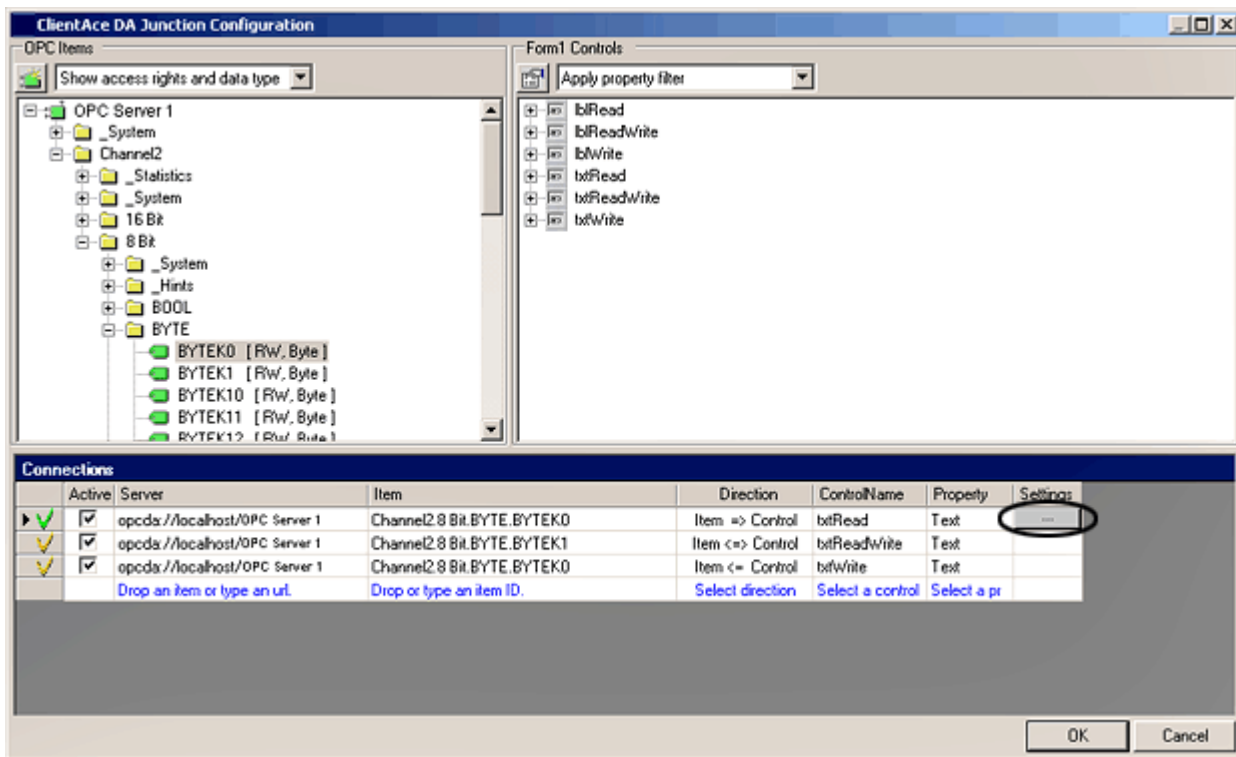
3. In the **Connection Settings window**, select the **General tab.**

4. Modify the value in the **Update Rate field** (in milliseconds).

5. Click **OK**.
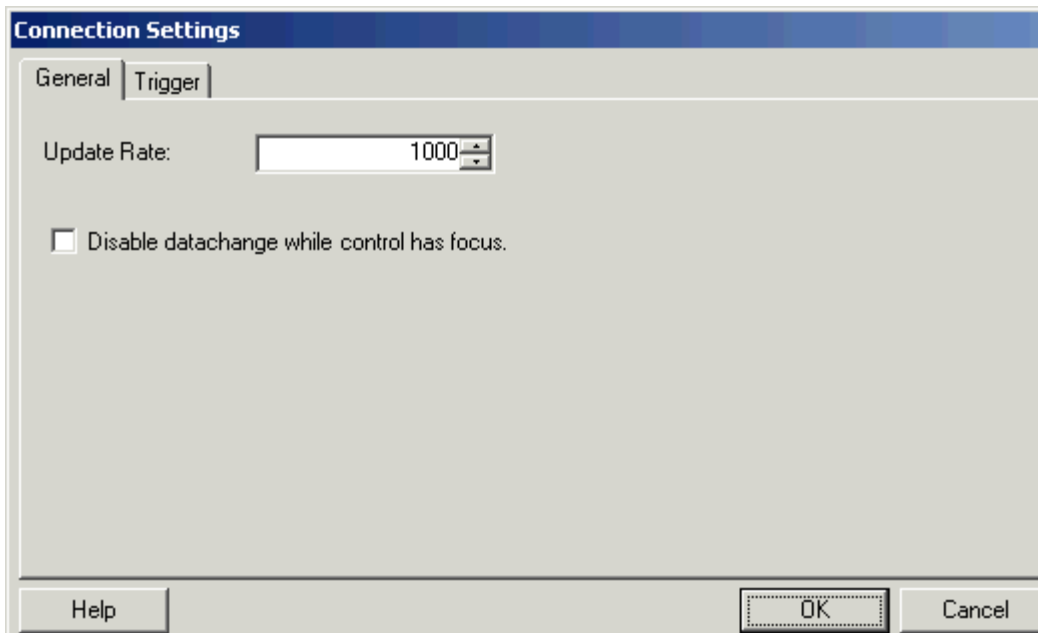
Disable DataChange while Control Has Focus

Disable datachange while control has focus allows you to change a value in the control without it being overwritten

by a change from the OPC Server.

Step1:

1. Launch the **Configuration window** by clicking on the **ClientAceConfiguration ellipses button**.
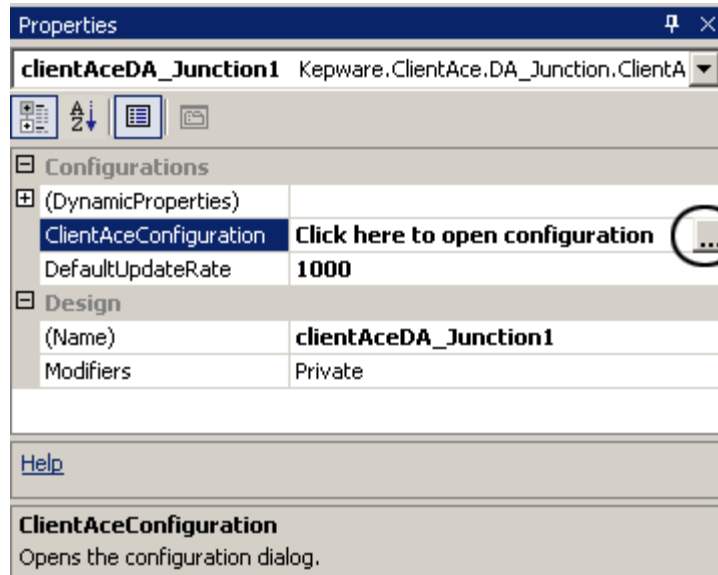
2. In the **Settings column**, choose the ellipses next to the item whose properties are to be changed.
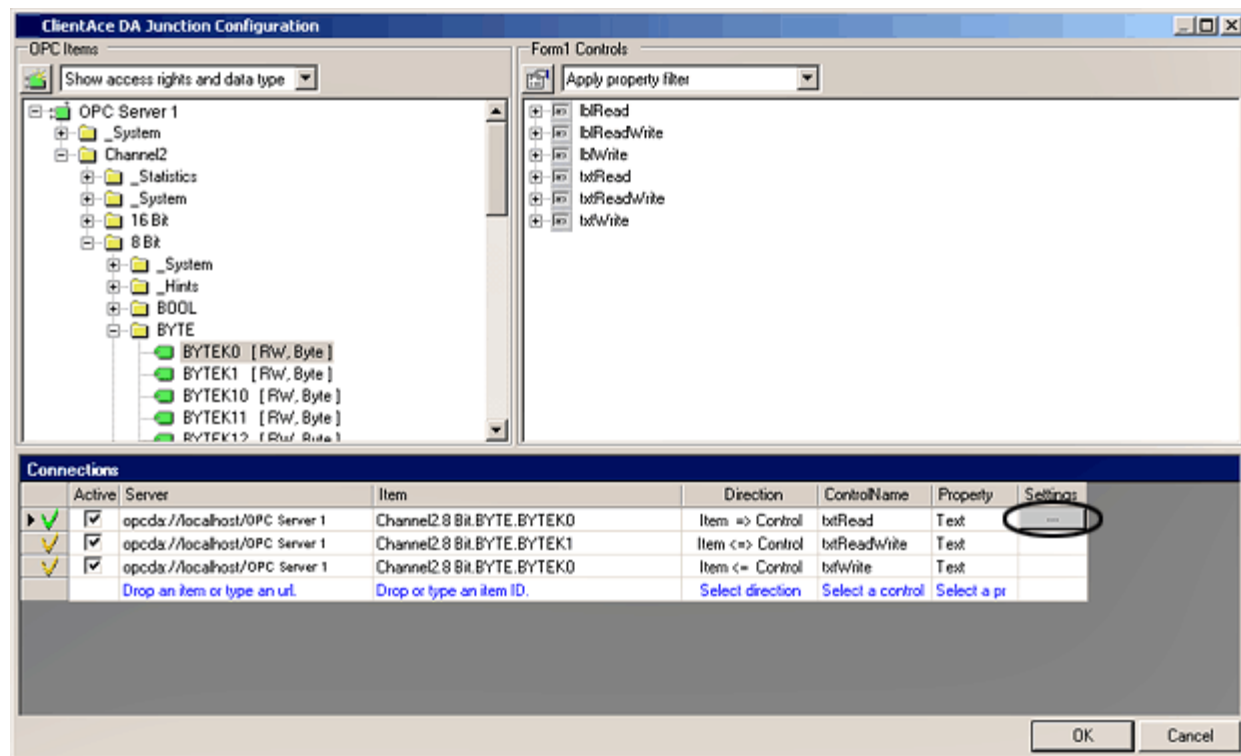
3. In the **Connection Settings window**, select the **General tab**.

4. Click the checkbox for **Disable datachange while control has focus**.
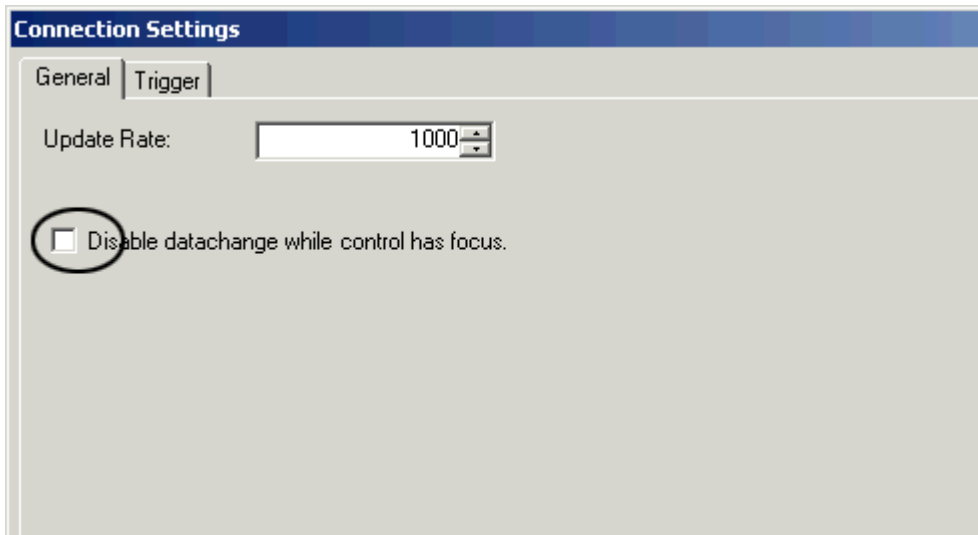
5. Click **OK** at the bottom of the Connection Settings window.

Note: The selected control is now set for the **Data Update Pause** when it has focus.

Data Types Description

Data Type	Description
Boolean	Single bit
Word	Unsigned 16 bit value bit 0 is the low bit bit 15 is the high bit
Short	Signed 16 bit value bit 0 is the low bit bit 14 is the high bit bit 15 is the sign bit
DWord	Unsigned 32 bit value bit 0 is the low bit bit 31 is the high bit
Long	Signed 32 bit value bit 0 is the low bit bit 30 is the high bit bit 31 is the sign bit
Float	32 bit floating point value bit 0 is the low bit bit 31 is the high bit
Double	64 bit floating point value bit 0 is the low bit bit 63 is the high bit
String	Typically null terminated, null padded or blank padded ASCII string

Additional ClientAce .NET Controls

ServerBrowser Control
ItemBrowser Control
ChannelSettings Control
ServerState Control

ServerBrowser Control

The ServerBrowser control provides the functionality to browse OPC Data Access servers on local and remote machines.

Adding the Control to the Visual Studio Project

Important: All referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, as this will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.

1. Open a new or existing project in Visual Studio.

2. Verify that all of the ClientAce controls have been added to the **Visual Studio Environment**. In **Visual Studio**, the **Toolbox** should include the controls shown below. To add controls to the Toolbox, see **Missing Controls**.
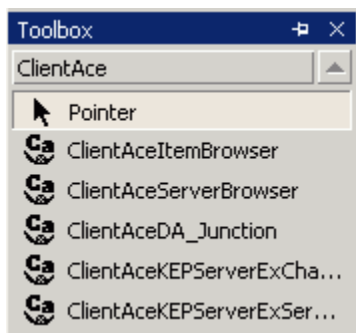
3. To **add a control**, drag it from the Toolbox and drop it onto a **form**.
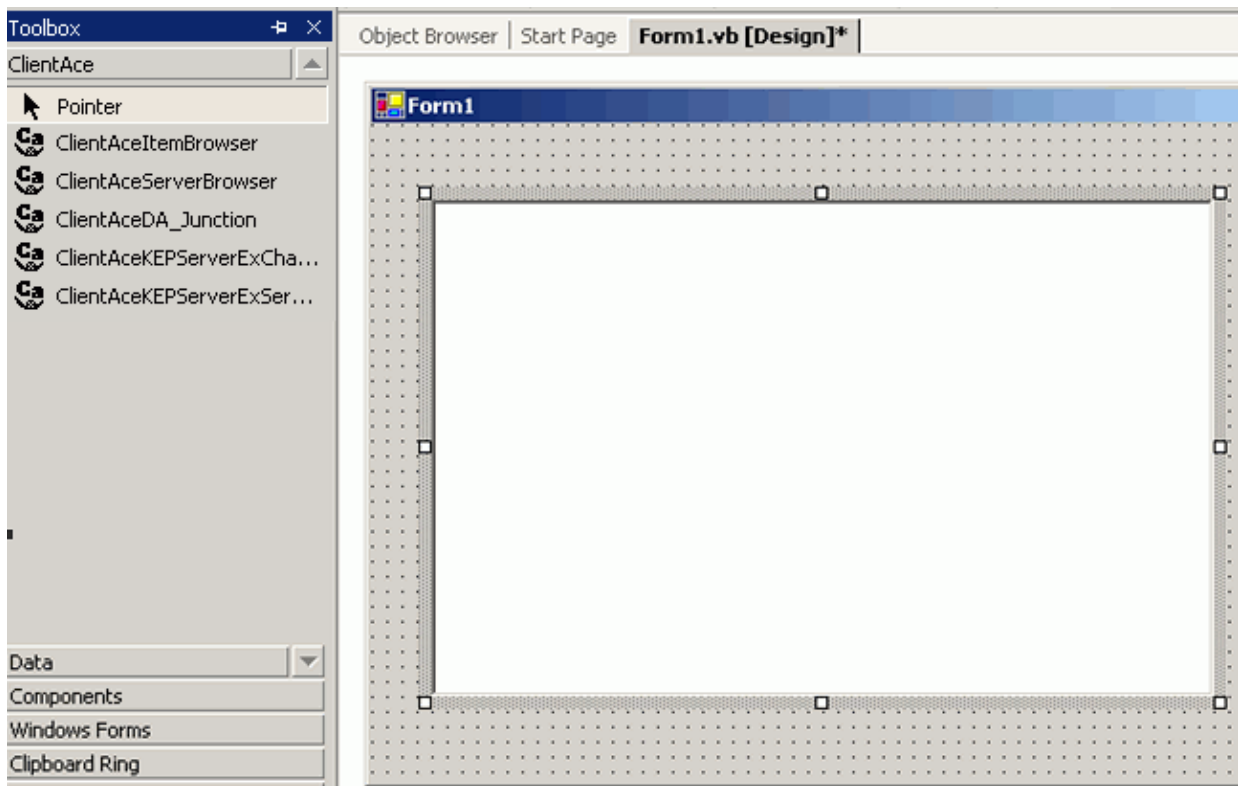
The ServerBrowser Control at Runtime

At Runtime, the ServerBrowser control looks like this:
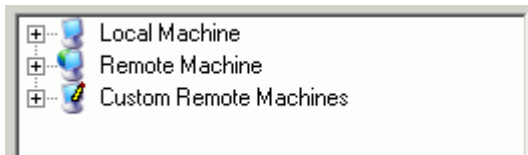
Local Machine

Click on the + to expand the **Local Machine** and display the **servers**. Click on a server to highlight it. For more infomation on using ClientAce API to connect to the server, refer to **Overview of ClientAce .NET API**.

Remote Machine

Click on the + to expand the **Remote Machine** and display the **servers**. Click on a server to highlight it. For more information on using ClientAce API to connect to the server, refer to **Overview of ClientAce .NET API**.

Note: The DCOM settings on the remote machine must be configured properly in order to access the servers on that machine.

Custom Remote Machines

Use Custom Remote Machines to custom define links to remote machines using either the IP address or machine name

of the PC that will be browsed. To define a custom link to a remote machine, perform the following steps:

1. Click on the + next to **Custom Remote Machines**.

2. Click on **<Add Node>** and then press **F2**.
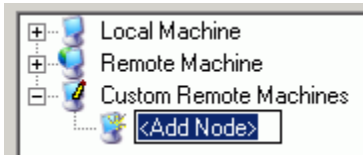
3. Type the **IP address or machine name** of the remote PC that will be browsed, and press **ENTER**.

4. A **link** pointing to the remote machine has been created. Click on the + next to the remote machine IP address or name to display the **servers** on the remote machine.

5. Click on a server to highlight it. To use the ClientAce API to connect to the server, refer to **Overview of ClientAce .NET API** for more information.
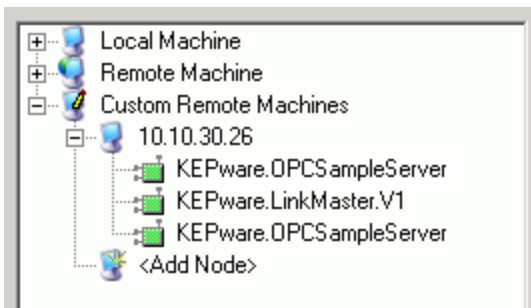
6. In this example, the remote machine 10.10.30.26 has been defined as a custom link.

Note: Once a Custom Remote Machine is created, the link is saved by the application. The next time the application is opened, the Custom Remote Machine will be available and accessible. Please note, however, that the Custom Remote Machine is associated only with the application that it was created for originally. For example: If a new application is created, the Custom Remote Machines created for other applications/projects will not be available for browsing. This means that a new Custom Remote Machine link would need to be created for the new application/project.

ItemBrowser Control

The **ItemBrowser** control provides the functionality to browse tags in an OPC Data Access server on local or remote machines.

Adding the Control to the Visual Studio Project

All referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, because it will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.

1. Open a new or existing project in Visual Studio.

2. Verify that all of the ClientAce controls have been added to the Visual Studio Environment. To add controls to the toolbox, refer to **Adding Controls to the Visual Studio Environment**.
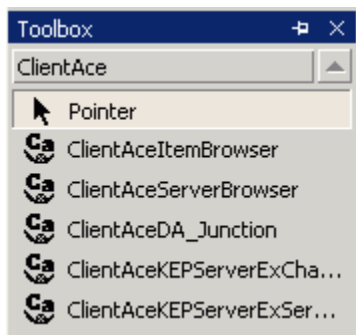
3. To add a control, drag it from the Toolbox and drop it onto a form.
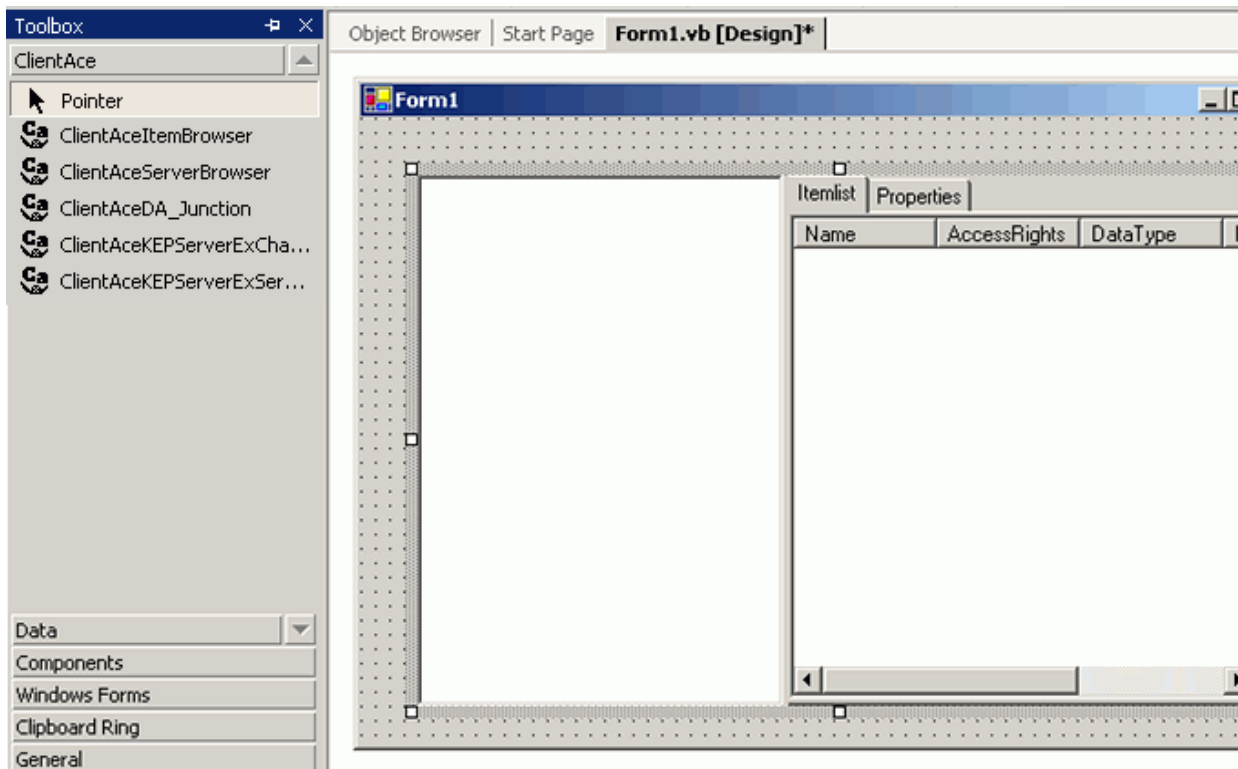
The ItemBrowser Control at Runtime

At Runtime, the ItemBrowser control looks like the following:

1. The blank left pane indicates that no servers have been added. To add a server, right-click in the left pane and select **Add Server** from the context menu.
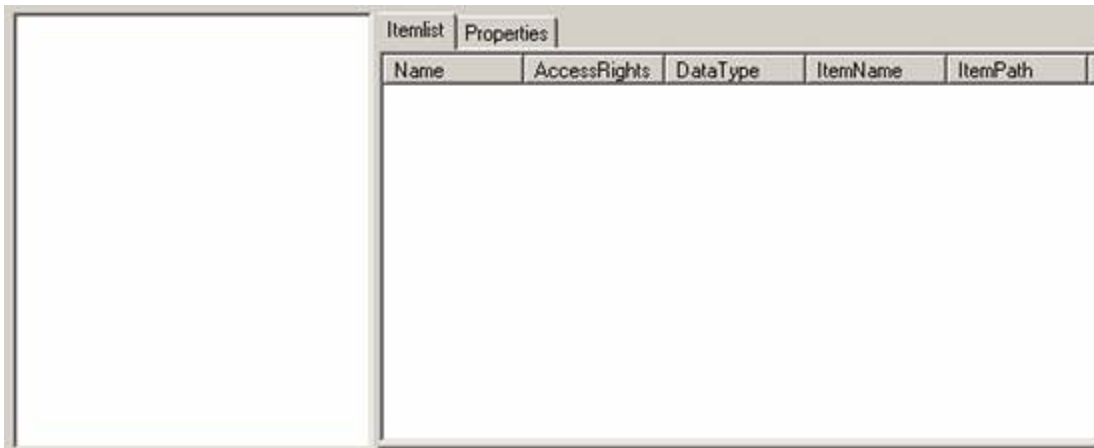
2. Next, **add an OPC server** using either the **Server Browser** or **OPC DA tabs**. To add a server using the **Server Browser tab**, see **ServerBrowser Control**. To add a server using the **OPC DA tab**, perform the following steps.
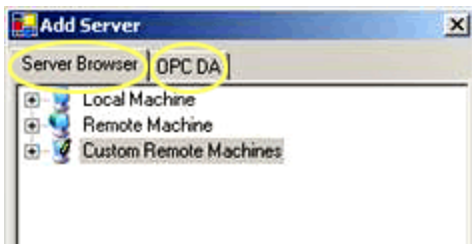
Note: When designing an application, it is best to synchronize the **ItemBrowser contro**l with the **ServerBrowser control**. You would not want to connect to a particular server using the ServerBrowser before adding tags of a different server using the ItemBrowser. For more information, refer to **ServerBrowser Control**.

3. Click the OPC DA tab and fill in the required details of the OPC server that will be connected to.
   Hostname: Enter any of the following: IP address, machine name, or localhost.
   ProgID: Enter the exact ProgID of the server.
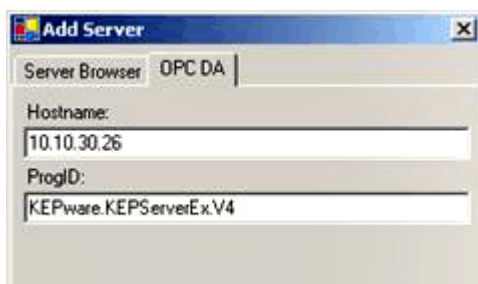
4. When finished, click **OK**. The chosen server can be found in the left pane of the **ItemBrowser window**. In the example shown below, server 10.10.30.26 has been added.
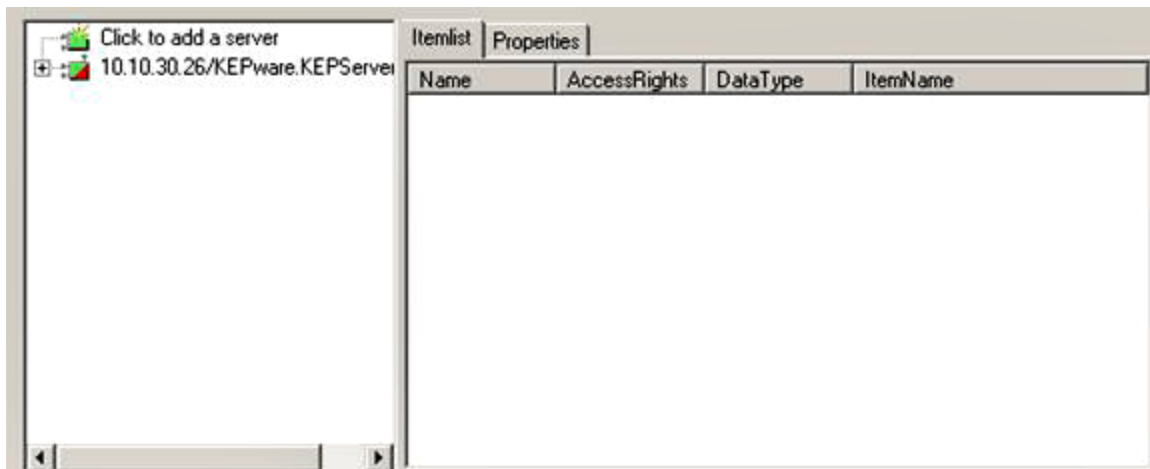
5. To expand the added server, click on the + next to the server name or IP address.

6. Select the channel by clicking on the + next to it.

7. Click on the tag group. The tags for that group will be displayed in the Itemlist tab in the right pane. The screenshot below shows the Device_1 group selected from Channel_1 in the 10.10.30.26 server. The four tags for the Device_1 group are shown in the Itemlist tab in the right pane.
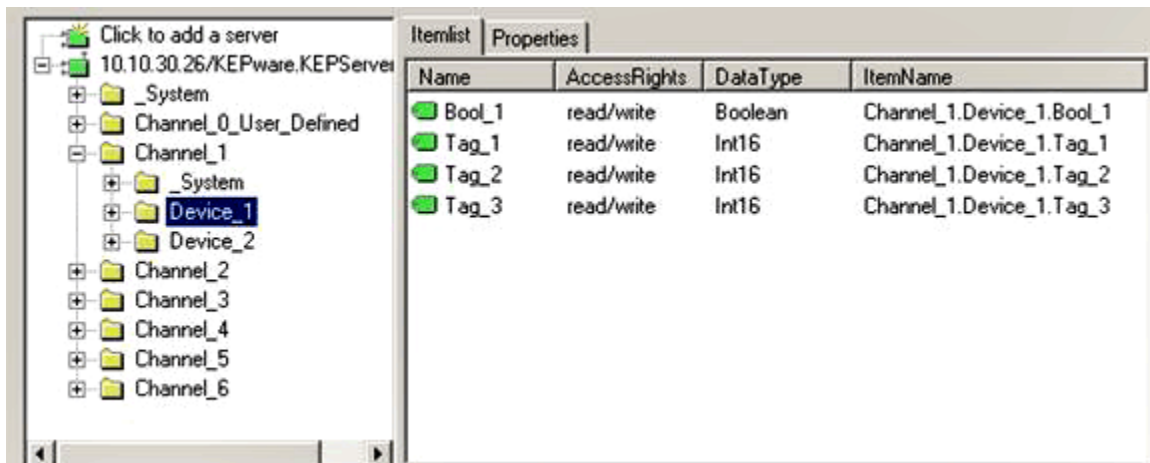
Note: The tags that are browsable in the ItemBrowser control can be selected and monitored by the programming code. To view the properties of a tag, select the tag and then click the Properties tab.
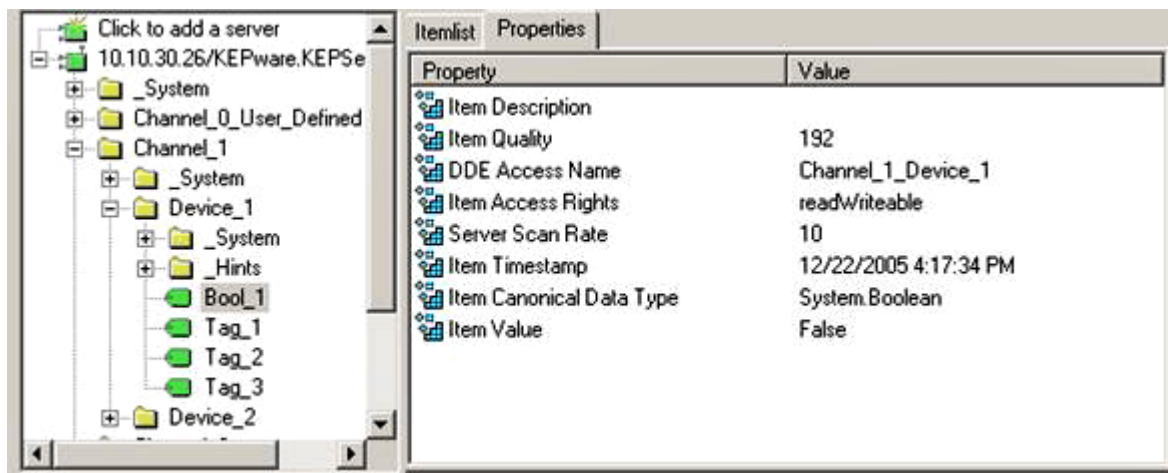
ChannelSettings Control

The **ChannelSettings** control provides the functionality to view and make certain changes to the properties of a channel in an OPC server provided by KEPware Technologies.

Note: If there are multiple KEPServerEX OPC servers installed on the local machine, the ChannelSettings control retrieves the channel properties of the server that was installed most recently.
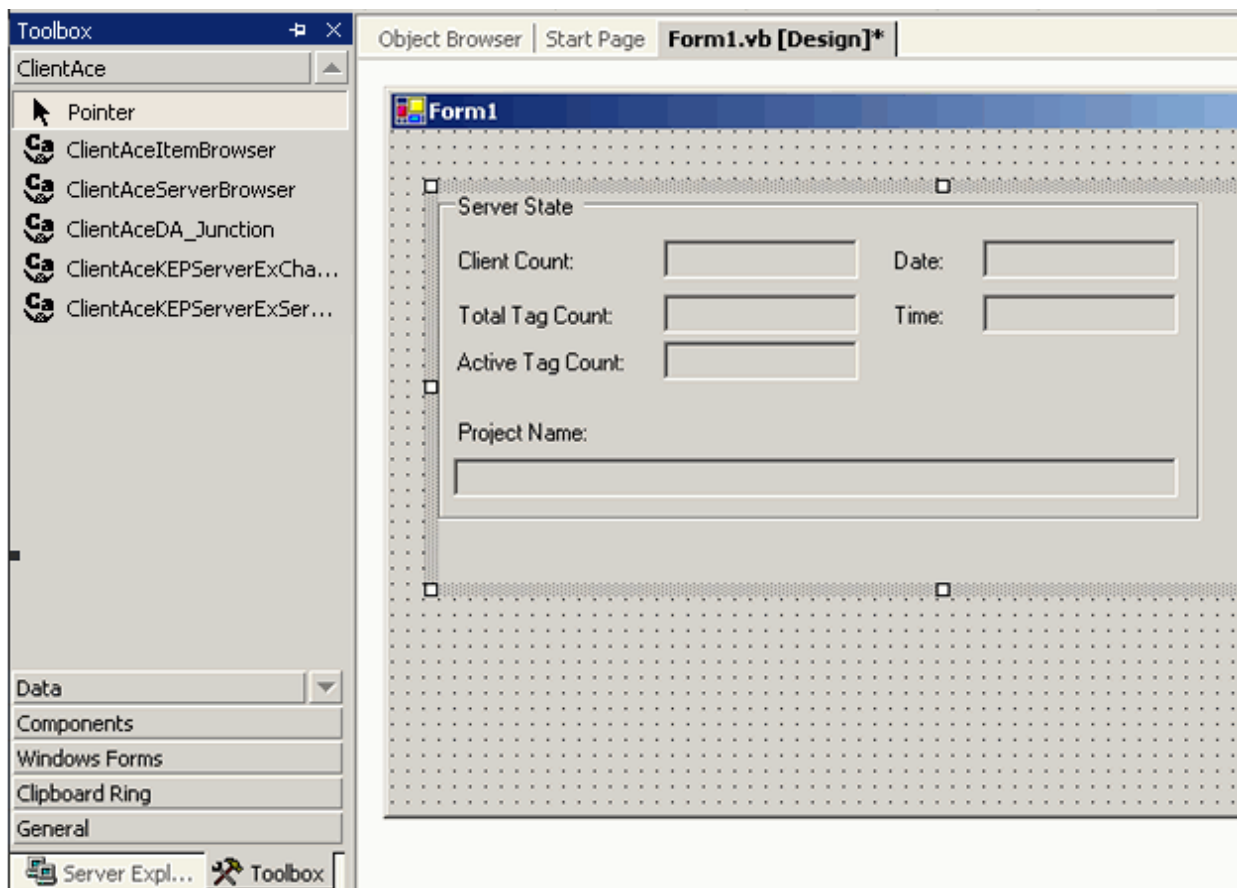
Adding the Control to the Visual Studio Project

Remember that all referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, because it will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.

1. Open a **new or existing project (solution)** in Visual Studio.

2. Verify that all of the ClientAce controls have been added to the Visual Studio Environment. In Visual Studio, the Toolbox should include the controls shown below. For more information on adding controls to the Toolbox, see **Adding Controls to the Visual Studio Environment.**
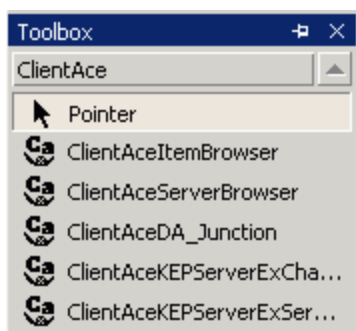
3. To add a control, drag it from the Toolbox and drop it onto a **form**. The image below shows the ChannelSettings control being added to a form.
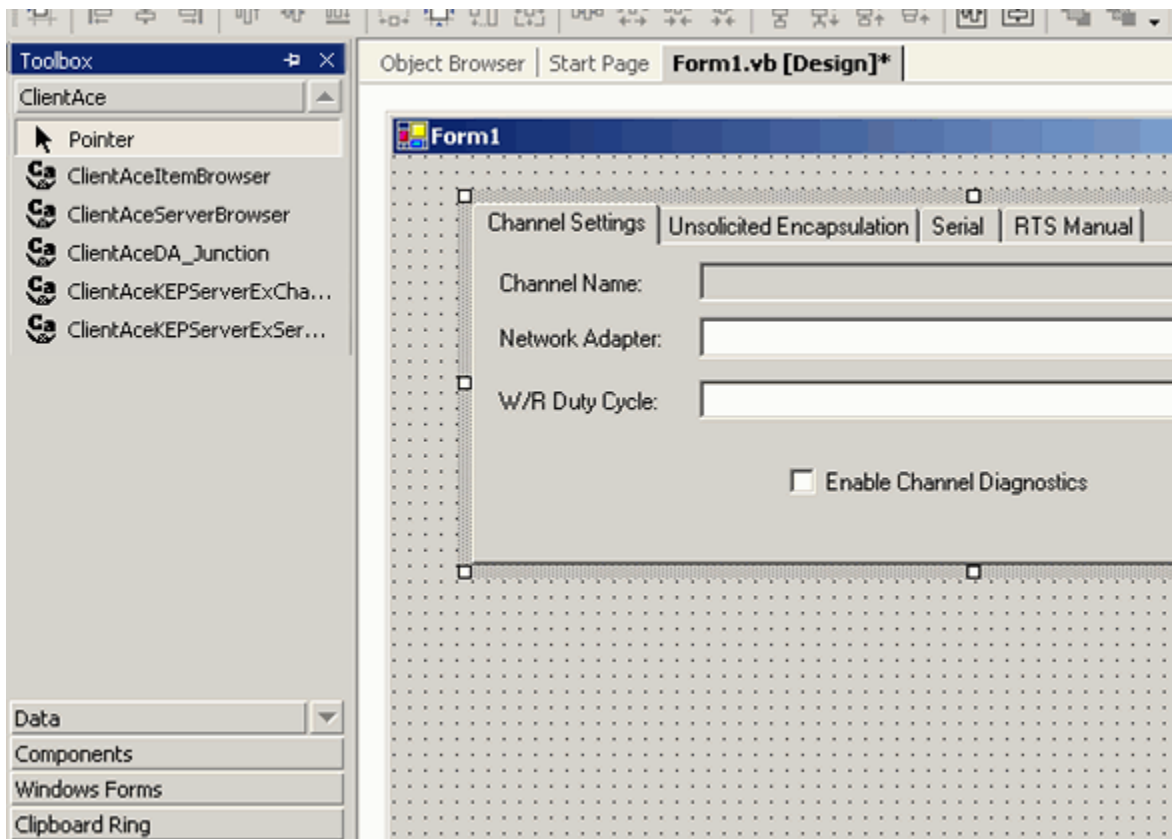
The ChannelSettings Control at Runtime
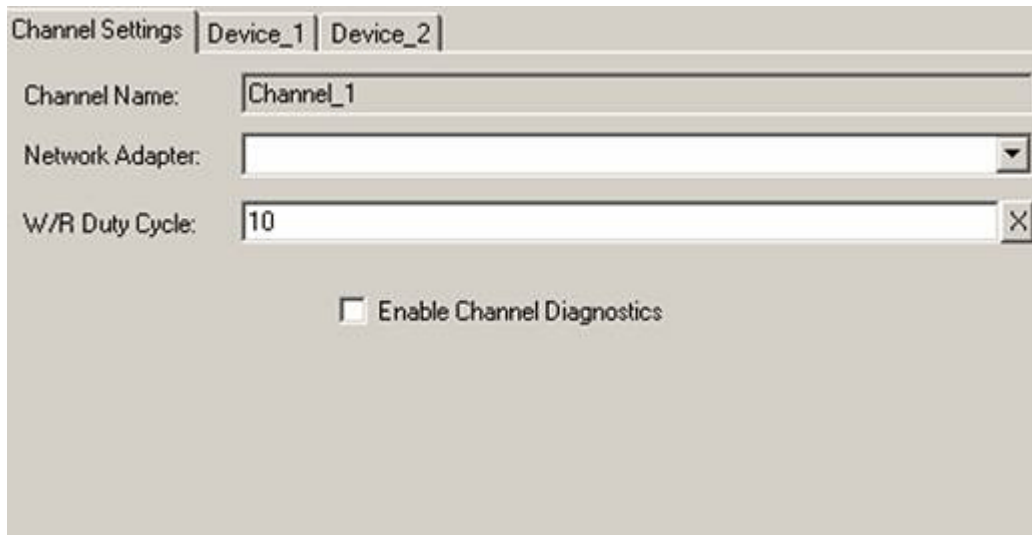
Remember that the control will have different tabs depending upon the type of channel (either serial or ethernet) to which the control links.
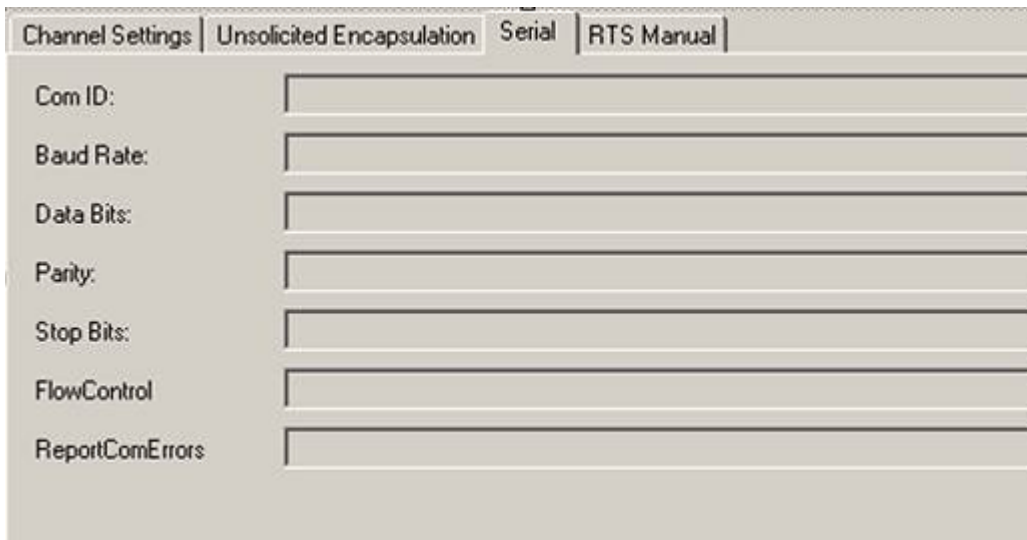
To link the ChannelSettings control to a specific channel, perform the following steps:
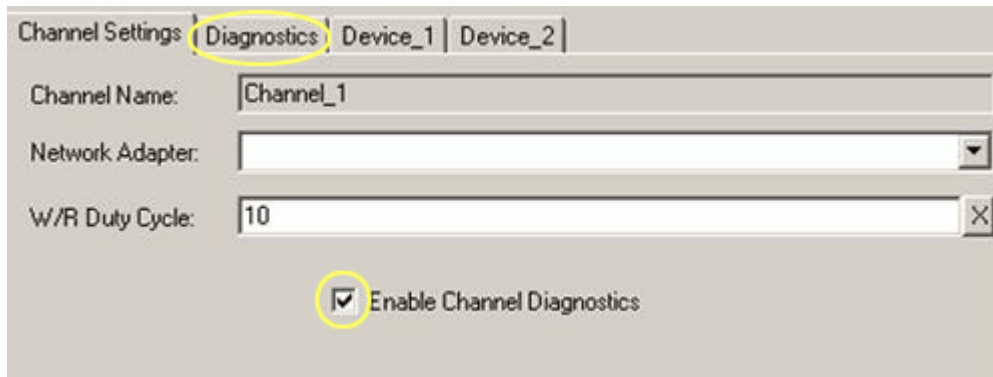
1. Right-click on the ChannelSettings control and select **Properties**.

2. Click on **ChannelName** and enter **Channel_1**. In this example, Channel_1 is used because that node name is

present in the sample KEPServerEX OPC project.

The **Channel Settings** tab displays the channel properties. If the channel used a network adapter, it would be listed in the Network Adapter field. Values in the Network Adapter field and W/R Duty Cycle field can be modified as needed. The **Enable Channel Diagnostics checkbox** is used to display diagnostics information in a separate Diagnostics tab, as shown in the following screenshots.

The **Device_1** and **Device_2** tabs display the properties of the two devices configured under the channel. If more devices were configured, the window would display a tab for each. Although the Device Properties are displayed, they cannot be modified in this window.

ServerState Control

The **ServerState** control provides the functionality to view the properties of the project of an OPC server provided by KEPware Technologies.

Note: If there are multiple KEPServerEX OPC servers installed on the local machine, the ServerState control retrieves the project properties of the server that was installed most recently.

Adding the Control to the Visual Studio Project

Important: All referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, as this will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.

1. Open a new or existing project in Visual Studio.

2. Verify that all of the ClientAce controls have been added to the **Visual Studio Environment**. In **Visual Studio**, the **Toolbox** should include the controls shown below. To add controls to the Toolbox, refer to **Missing Controls** .
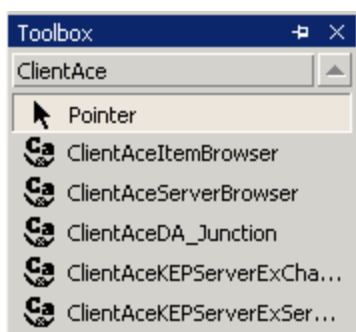
3. To **add a control**, drag it from the **Toolbox** and drop it onto a **form**.

The Control at Runtime
At Runtime, the ServerState control looks like this:

Note: Initially, the tag count displayed in the **Total Tag Count** and **Active Tag Count** fields is 6, to account for the six state properties that are displayed: Client Count, Total Tag Count, Active Tag Count, Date, Time, and Project Name.

Demo Mode

Unless ClientAce is licensed and all runtime applications built with ClientAce .NET controls have been signed, the applications will run in demo mode for 1 hour. After the demo period expires, another demonstration period can be started by restarting the application. After ClientAce is licensed and the runtime applications built with ClientAce .NET controls are signed, the applications will run in unlimited runtime operation.

See Also:

Licensing ClientAce
Signing Your Client Application

Licensing ClientAce

ClientAce .NET controls on the development PC must be licensed in order for **custom client applications to be signed** for unlimited runtime operation. If the applications are not licensed, they will run in **demo mode**.

Note: For all licensing questions, please contact Kepware Technologies at support@kepware.com or (888) 537-9273 ext. 211.

To License ClientAce:
1. Go to **Start** | **Programs** | **Kepware Products** | **ClientAce** | **License ClientAce**.
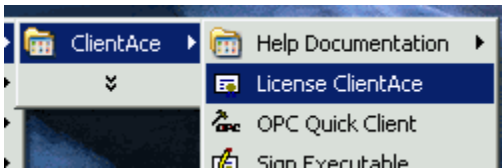
2. In the Kepware ClientAce License dialog, click **Acquire License**.
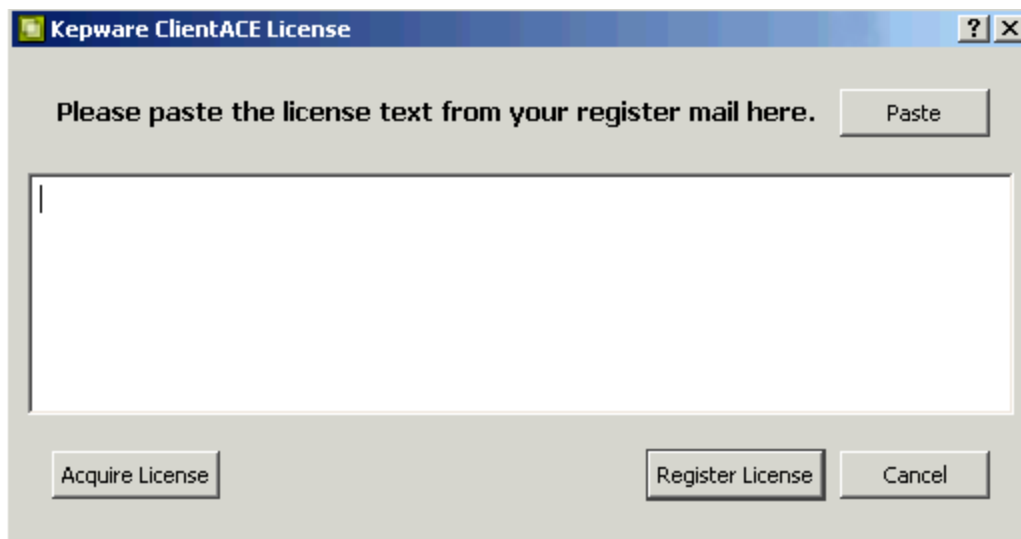
3. The **Registration Information dialog** is displayed. As the **Name** and **Company** fields get types, the **License Information field** will be populated with the licensing information needed by Kepware Technologies.
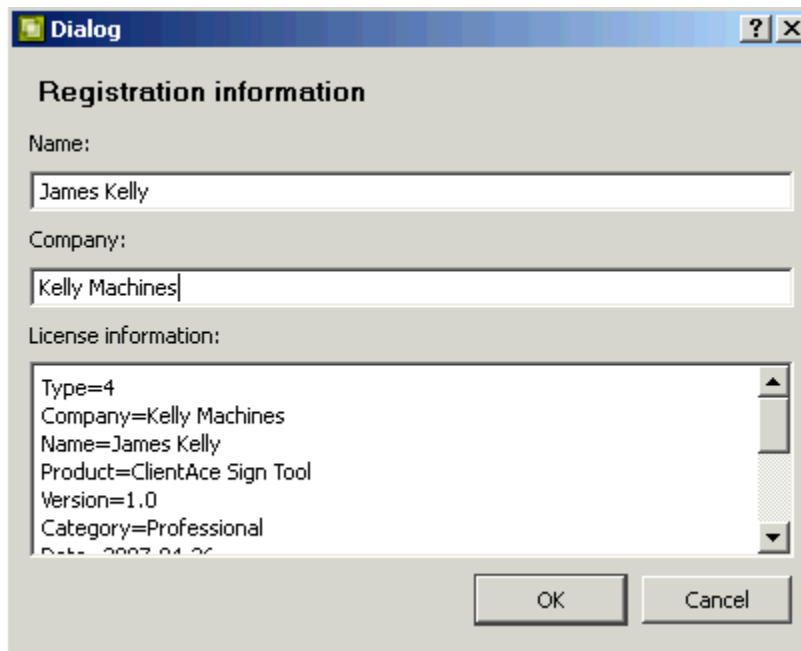
4. Click **OK**. An email message window from your email client application will be displayed. To send the message to Kepware Technologies, click **Send**.

5. Kepware Technologies will then send an email reply containing the **licensing code**. Copy the code into the **Kepware ClientAce License dialog** window, as shown below.
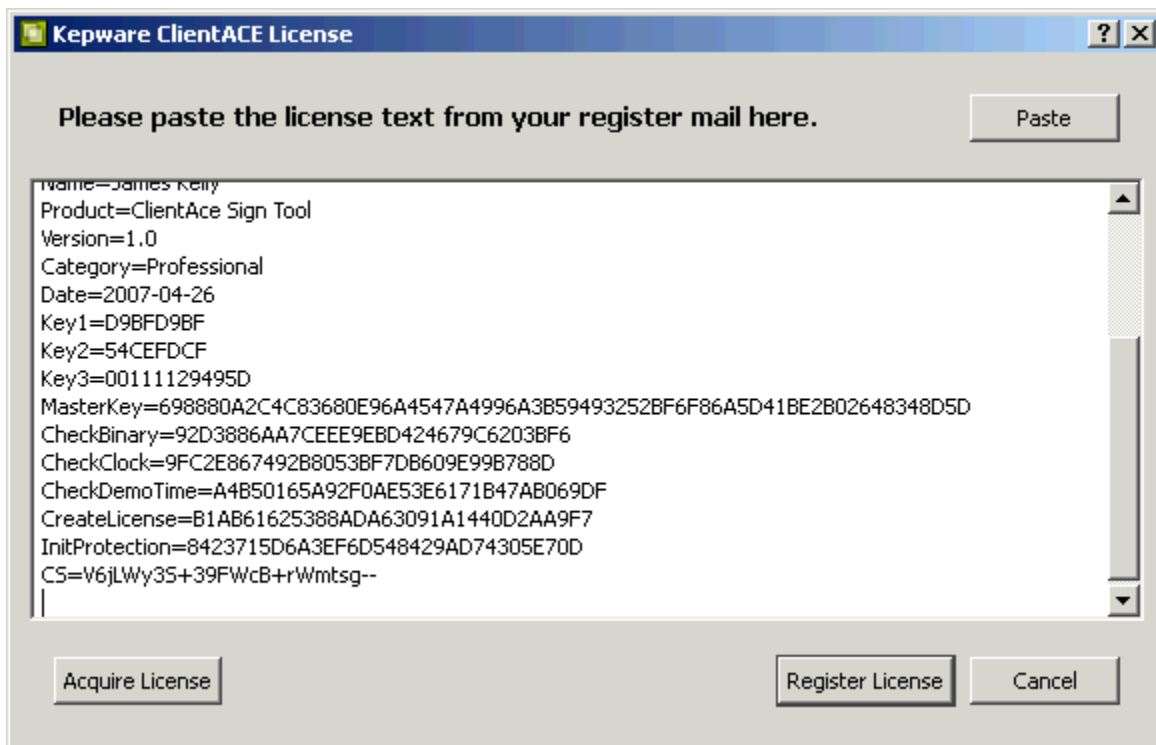
6. Click **Register License**. After the **confirmation message** is displayed, click **OK** to close the dialog.
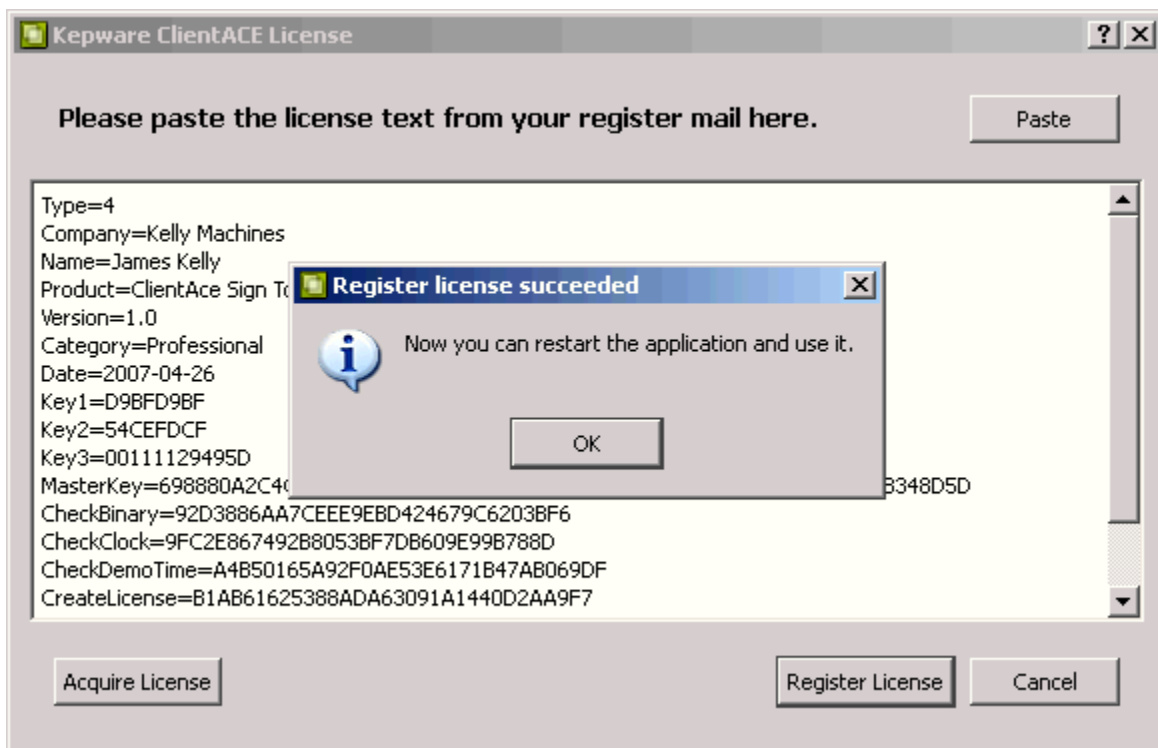
Now that ClientAce is licensed, the custom client applications that have been built may now be **signed**.

Signing Your Client Application

Applications created using a ClientAce .NET controls must be signed before they will run for unlimited runtime operation. If the application is not signed, it will run in **demo mode**.

Note: ClientAce must be licensed from Kepware Technologies before applications can be signed. For more information, refer to **Licensing ClientAce**.

To Sign the Custom Client Application Using the Visual Studio Sign Add-in:
Open the project that needs to be signed, and click the **Sign** icon in the toolbar. This will tag the project's executable file to be signed whenever the project is built.

Note: The license file (*.lic) is saved in the same folder as the executable file.
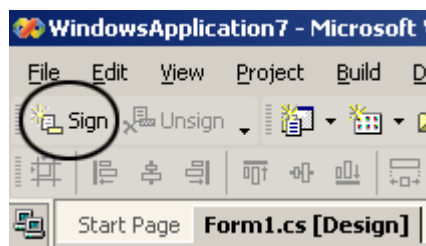
The project is now set to be signed automatically every time the project is built.

Manually Signing Your Custom Client Application
If the VS Add-in tool was not chosen to sign the custom client application, follow these steps to sign it manually.

Note: If the application was signed manually, the steps must be repeated to sign the application every time the project is built.

1. Select **Start** | **Programs** | **Kepware Products**| **ClientAce**| **Sign Executable**.

2. In the **Signing GUI dialog**, click the **ellipses** to browse for your application's **executable file**.
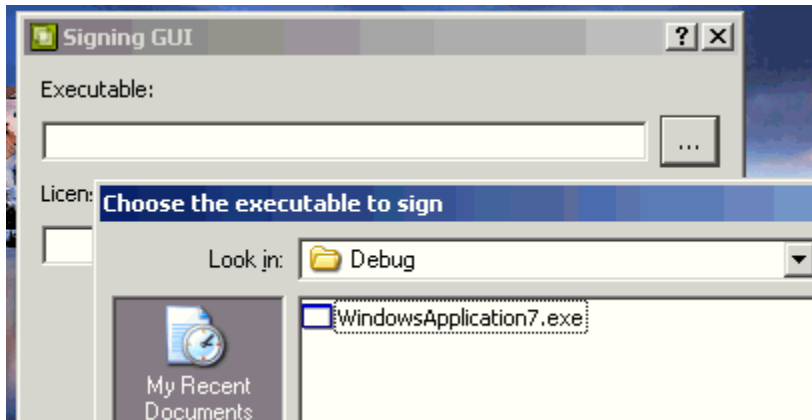
3. When choosing the executable file, the signed license code is displayed in the **License File field**. Note that the license file (*.lic) is saved in the same folder as the executable file.

4. Click **OK** to save and exit.

Deploying Your Client Application

Depending on the ClientAce features being used by the application, one or more of the following files may be required for the application to run properly:

- Kepware.ClientAce.Base.dll

- Kepware.ClientAce.BrowseControls.dll

- Kepware.ClientAce.Da_Junction.dll

- Kepware.ClientAce.KEPServerExControls.dll

- Kepware.ClientAce.OpcClient.dll

YourCustomClientAceApplication.exe
YourCustomClientAceApplication.lic

These files will be located in the output build directory created by Visual Studio for the project. When deploying the client application, these files must be installed in the same location as the custom client executable files.

.NET Framework Requirements

When deploying custom client applications created using ClientAce, it is required that .NET Framework 1.1 be installed on the PC that the client will deploy on. If the client application utilizes functionality from a version of the .NET Framework that is higher then the .NET 1.1 Framework, then that version also will be required to be installed.

To check if .NET Framework is installed:

1. Click **Start** on the Windows desktop.

2. Select the **Control Panel**.

3. Double-click the **Add or Remove Programs** icon.

4. Next, scroll through the list of applications. If Microsoft .NET Framework 1.1 is listed, the version required by ClientAce is already installed and does not need to be installed again.

To obtain versions of the .NET Framework, click **Start** on the Windows desktop and select **Windows Update**.

Note: The actual ClientAce install does not need to be installed on the destination computer in order for the custom

ClientAce application to work.

See Also:
System and Application Requirements
Licensing ClientAce
Signing Your Client Application.

Troubleshooting

Click on the following topics for descriptions of common troubleshooting problems.

Missing Controls
Referencing Controls
CoInitializeSecurity
Visual Studio 2005 LoaderLock Exception
Removing Blank Toolbar Options after Uninstalling ClientAce (VS 2005)
ASP .NET Development Incompatibility

Missing Controls

The following controls are typically added to the system's Visual Studio Environment automatically during the ClientAce installation process. If the Toolbox does not have any of the ClientAce controls, it is possible that the controls were unchecked during the ClientAce installation process.

ClientAce Controls (required):
- DA_Junction
- ServerBrowser
- ItemBrowser

Kepware-specific Controls (optional):
- ClientAceKEPServerEXChannelSettings
- ClientAceDEPServerEXServerState

To Add ClientAce Controls to the Visual Studio Environment:

Important: All referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, as this will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.
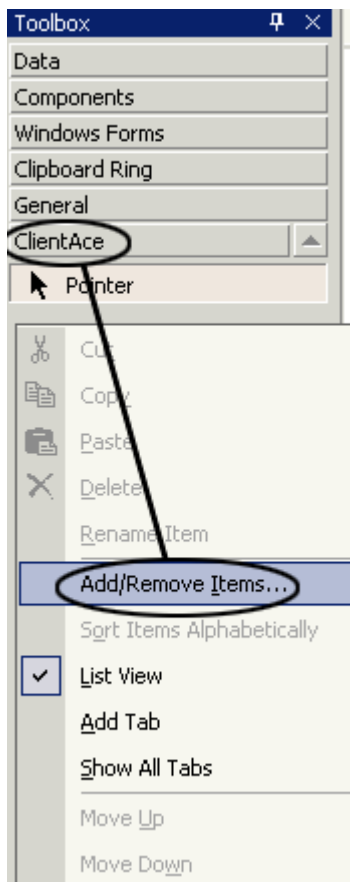
1. Open a **new C#** or **Visual Basic project** using the Visual Studio .Net application.

2. Right-click anywhere on the **ToolBox window** and select **Add Tab**.
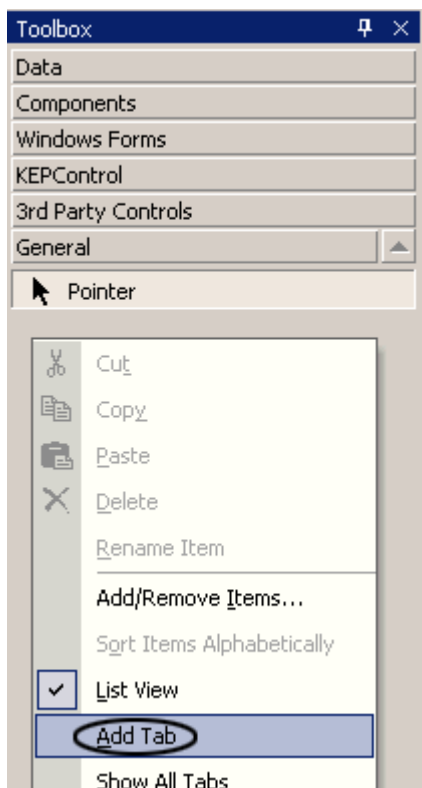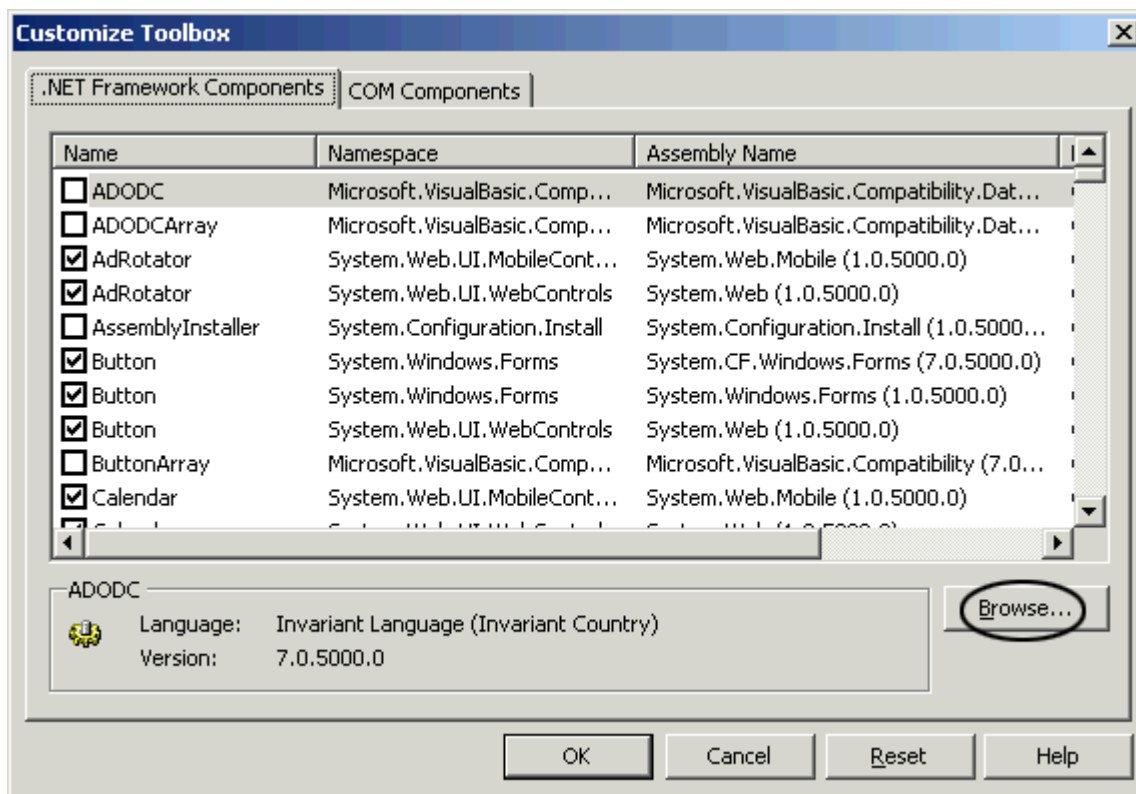
3. Enter "**ClientAce**" in the empty box. This creates the ClientAce tab.
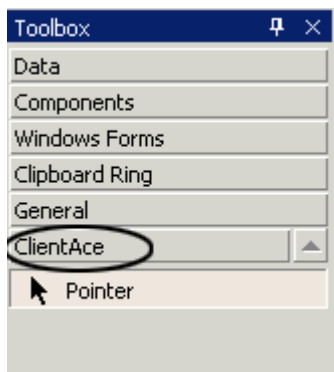
4. Right-click anywhere on the ClientAce tab and select **Add/Remove Items**. **Note:** In Visual Studio 2005, this will be **Choose Items**.

5. In the **Customize Toolbox window,** click on the **Browse**. Navigate to the directory where the **ClientAce.dll** files are stored.

6. First, click to select the .dll file that contains the controls yet to be added. Then, click **Open** (or double-click the .dll file).

Kepware.ClientAce.DA_Junction.dll: DA Junction control
Kepware.ClientAce.BrowseControls.dll: ServerBrowser and ItemBrowser controls (see **Additional ClientAce Controls**)
Kepware.ClientAce.KEPServerExControls.dll: ChannelSettings and ServerState (see **Additional ClientAce Controls**)
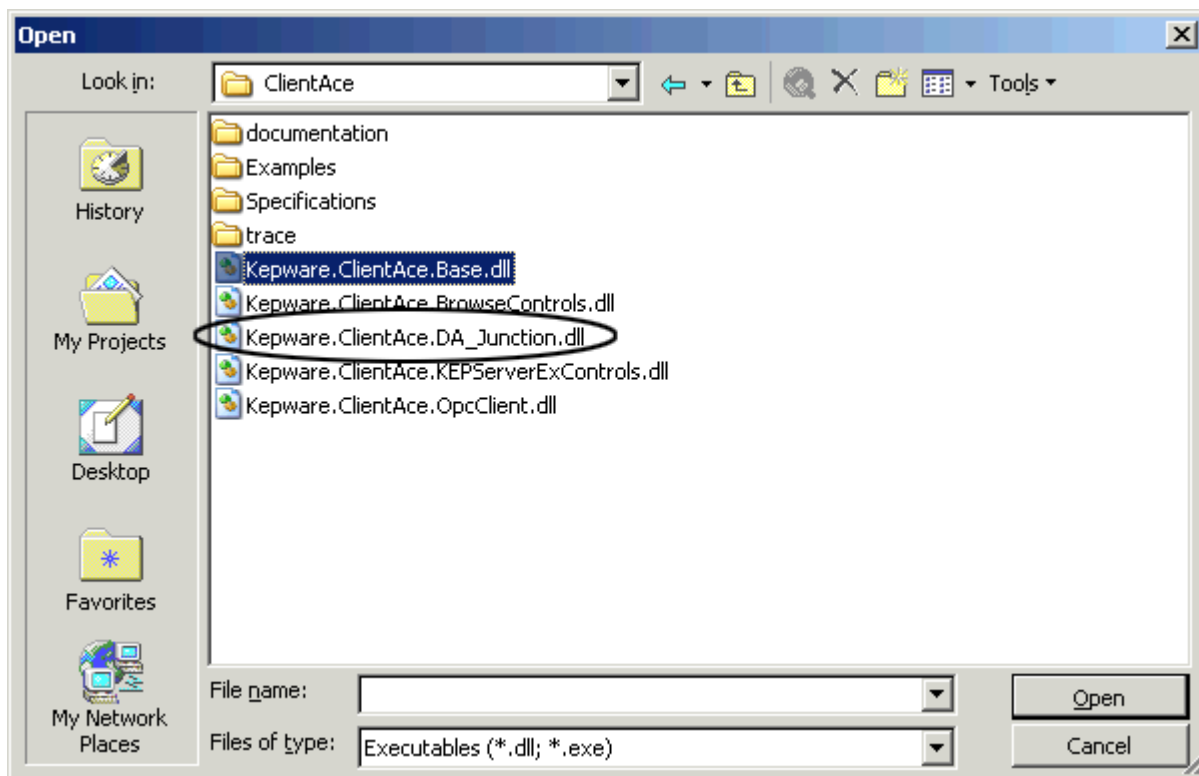
7. Selecting a .dll file displays the Customize Toolbox window. In the example shown below, the ClientACE.DA_Junction library is now checked for inclusion.
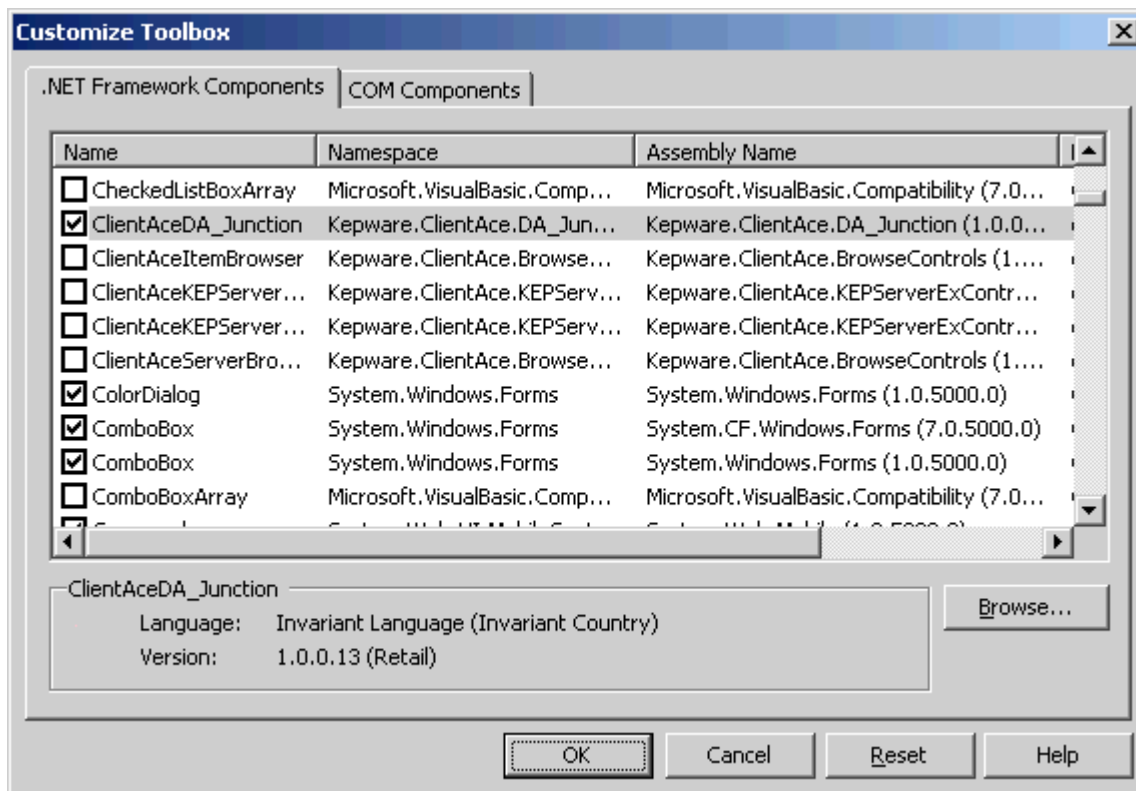
8. To add other controls, click on **Browse** and select another .dll file. Repeat until all the control files (all the .dll files) have been added to the **Customize Toolbox for inclusion**.

9. Click **OK** at the bottom of the Customize Toolbox window. The Toolbox will display all controls that have been added.

Note: To display the applicable references in the Solution Explorer, select **View | Solution Explorer**. Controls that have been added to the Visual Studio Environment can also be added to the Visual Studio project by dragging them from the **Toolbox | ClientAce tab** onto the form. For more information, refer to **Additional ClientAce Controls**.

Referencing Controls

All referenced controls must be on the local drive. Assemblies that are located on a network drive should not be referenced, as this will cause the Visual Studio error "Unable to cast object of type <type> to <type>." This is a limitation of the Microsoft .NET development environment.

CoInitializeSecurity

The ClientAce application must set its security credentials such that an OPC server has the privilege to send OnDataChange/OnServerShutDown notifications to the client. In order to set the security credentials, a ClientAce application must set the security level using CoInitializeSecurity during the initialization of the application.

In order to call CoInitializeSecurity in the ClientAce application, see the VB and C# examples shown below.

Visual Basic Example

```
' .Net library for Interoperability

Imports System.Runtime.InteropServices

' declaring the enums for the CoInitializeSecurity call

Public Enum RpcImpLevel

    E_Default = 0      E_Anonymous = 1

    E_Identify = 2     E_Impersonate = 3

    E_Delegate = 4            End Enum

Public Enum EoAuthnCap

    E_None = &H0

    E_MutualAuth = &H1

    E_StaticCloaking = &H20

    E_DynamicCloaking = &H40

    E_AnyAuthority = &H80

    E_MakeFullSIC = &H100

    E_Default = &H800

    E_SecureRefs = &H2

    E_AccessControl = &H4

    E_AppID = &H8

    E_Dynamic = &H10
```

```vb
    E_RequireFullSIC = &H200

    E_AutoImpersonate = &H400

    E_NoCustomMarshal = &H2000

    E_DisableAAA = &H1000     End Enum

Public Enum RpcAuthnLevel

    E_Default = 0     E_None = 1

    E_Connect = 2     E_Call = 3

    E_Pkt = 4         E_PktIntegrity = 5

    E_PktPrivacy = 6          End Enum

'end of enums declared for the CoInitializeSecurity call
```

(Continued)

(VB example continuation)

```vb
Public Class Form1

    Inherits System.Windows.Forms.Form

    ' declare the CoInitializeSecurity signature within the class where it

    ' should be called (must be called before launching form

    Declare Function CoInitializeSecurity Lib "ole32.dll"

(

    ByVal pVoid As IntPtr, _

    ByVal cAuthSvc As Integer, ByVal asAuthSvcByVal As IntPtr, _

    ByVal pReserved1 As IntPtr, ByVal dwAuthnLevel As Integer, ByVal dwImpLevel
As Integer, _

    ByVal pAuthList As IntPtr, ByVal dwCapabilities As Integer, ByVal pReserved3
As IntPtr) As Integer

#Region " Windows Form Designer generated code "

    Public Sub New()

        MyBase.New()

        ' good place to call CoInitializeSecurity

        CoInitializeSecurity(IntPtr.Zero, -1, IntPtr.Zero, _

                        IntPtr.Zero, RpcAuthnLevel.E_None, _
```

```
                         RpcImpLevel.E_Impersonate, IntPtr.Zero, EoAuthnCap.
E_None, IntPtr.Zero)

      'This call is required by the Windows Form Designer.

      InitializeComponent()

      'Add any initialization after the InitializeComponent() call

   End Sub
```

C# Example

```csharp
// .net library required for interoperability

using System.Runtime.InteropServices;

// ******Enums required for CoInitializeSecurity call through C#.......//

public enum RpcImpLevel
   { Default    =   0,   Anonymous   =   1,

     Identify   =   2,   Impersonate   =   3,

     Delegate   =   4  }

public enum EoAuthnCap
   {  None = 0x00,

     MutualAuth = 0x01,

     StaticCloaking= 0x20,

     DynamicCloaking= 0x40,

     AnyAuthority= 0x80,

     MakeFullSIC= 0x100,

     Default= 0x800,

     SecureRefs= 0x02,

     AccessControl= 0x04,

     AppID= 0x08,

     Dynamic= 0x10,

     RequireFullSIC= 0x200,

     AutoImpersonate= 0x400,

     NoCustomMarshal= 0x2000,

     DisableAAA= 0x1000   }

public enum RpcAuthnLevel
```

```
  { Default =  0,    None   =  1,

    Connect =  2,    Call   =  3,

    Pkt     =  4,    PktIntegrity =  5,

    PktPrivacy  =  6  }
```
```
/*****************end of enum declarations for CoInitializeSecurity call******/
```

(Continued)

(C# example continuation)

```
namespace CSharpTestClient

{

public class Form1 : System.Windows.Forms.Form

    {  // Import the CoInitializeSecurity call from

    [DllImport("ole32.dll", CharSet = CharSet.Auto)]

    public static extern int CoInitializeSecurity( IntPtr pVoid, int

cAuthSvc,IntPtr asAuthSvc, IntPtr pReserved1, RpcAuthnLevel level, RpcImpLevel
impers,IntPtr pAuthList, EoAuthnCap dwCapabilities, IntPtr

        pReserved3 );

private Kepware.ClientAce.DA_Junction.ClientAceDA_Junction clientAceDA_Junction1;

    private System.Windows.Forms.TextBox textBox1;

    public Form1()

        {

        InitializeComponent();

}

        /// <summary>

        /// The main entry point for the application.

        /// </summary>

    [STAThread]

    static void Main()

        {

// call the CoInitializeSecurity right before Launching the Application

        CoInitializeSecurity( IntPtr.Zero, -1, IntPtr.Zero,
```

```
                    IntPtr.Zero,RpcAuthnLevel.None ,

RpcImpLevel.Impersonate,IntPtr.Zero,  EoAuthnCap.None,  IntPtr.Zero );

        Application.Run(new Form1());

        }

    }

}
```

## Visual Studio 2005 LoaderLock Exception

LoaderLock Exception

When developing an application with the ClientAce components, LoaderLock exception dialog may be encourntered when attempting to run within the context of the Visual Studio Debugger.
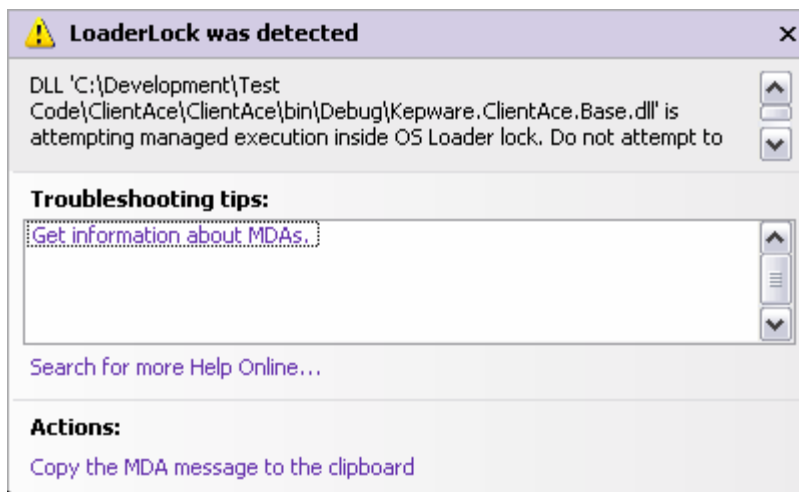
This warning occurs due to the use of Mixed (Native and Managed) Assemblies used by the ClientAce components. It is possible that the initialization of Mixed Assemblies could cause a deadlock in an application if the assemblies do not follow the strict requirements for initialization. ClientAce follows these rules, and this warning can be safely ignored.

Since Visual Studio may not properly start the application in the debugger after displaying this warning, it is recommended that the Managed Debug Assistant for the LoaderLock exception is disabled as follows:

1. Stop debugging.
2. Select **Debug** | **Exceptions**.
3. Expand the **Managed Debug Assistance** item.
4. Deselect the **Thrown** checkbox associated with the **LoaderLock** item.
5. Select **OK**.
6. Restart debugging.
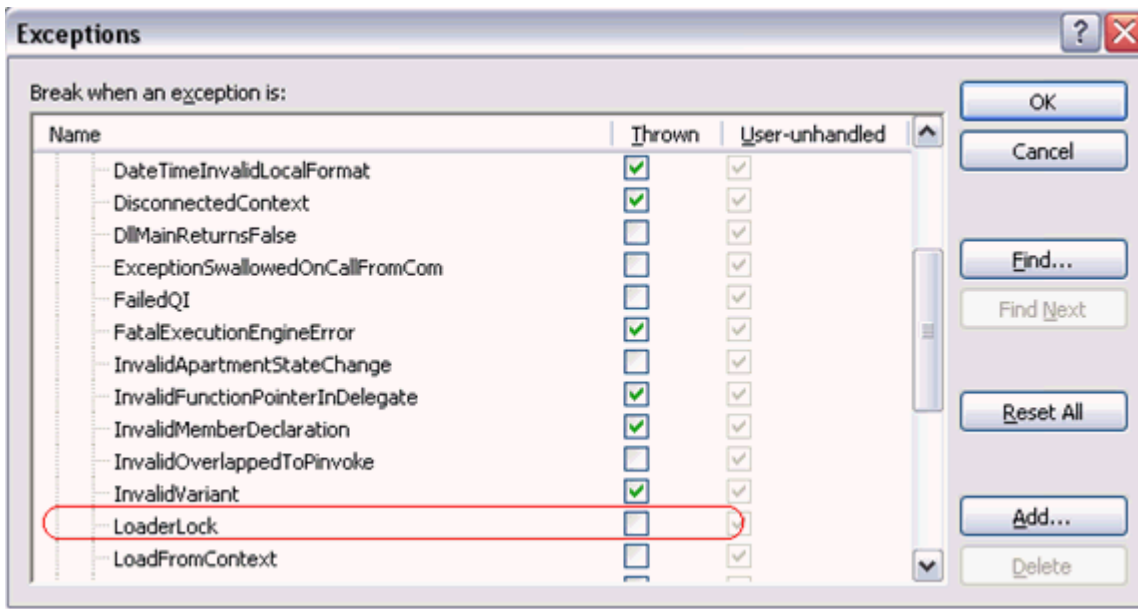
Removing Blank Toolbar Options after Uninstalling ClientAce (VS 2005)

If ClientAce is uninstalled, the Microsoft Visual Studio 2005 toolbar will have a blank space where the **Sign** and **Unsign** icons were. For more information, refer to **How to Sign an Application**.
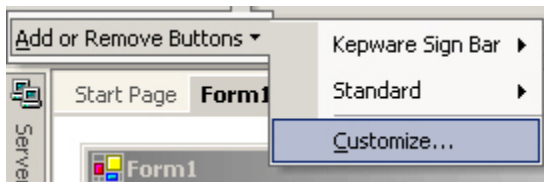
Note: This is only an issue with Visual Studio 2005, not VS 2003.

To remove the blank toolbar options from Visual Studio 2005 after uninstalling ClientAce:
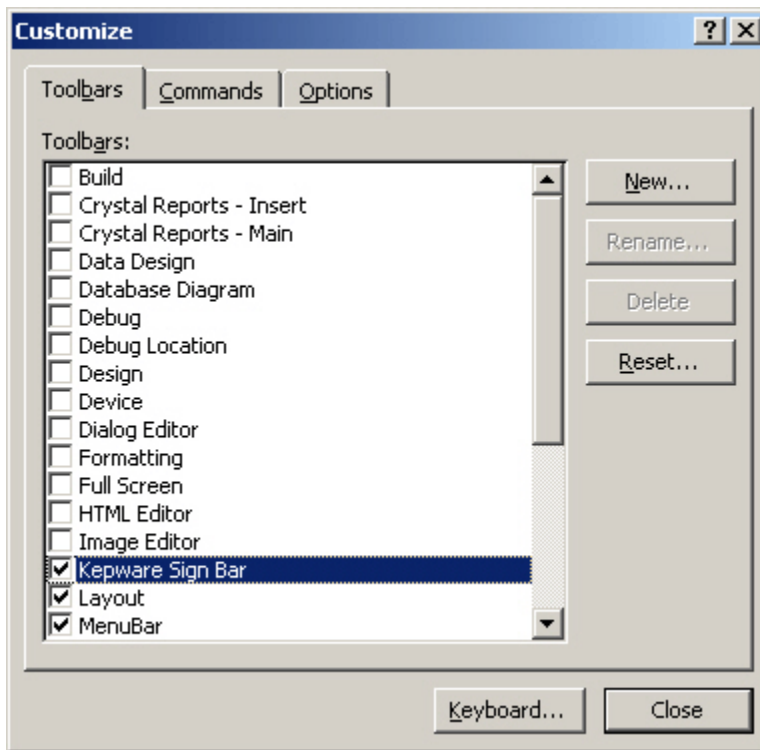1. In Visual Studio, click on the small arrow on the right edge of the blank toolbar option, then select **Add or Remove Buttons**.
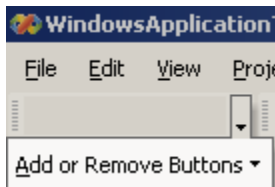
2. Select **Customize**.

3. In the **Toolbars tab**, scroll down to **Kepware Sign Bar**. Check Kepware Sign Bar, then click the **Delete** button.

ASP .NET Development Incompatibility

ClientAce cannot be used to develop ASP .NET applications. If ASP .NET OPC clients must be developed, please contact Kepware Technical Support.

Appendix 1 - ResultID Codes Enumeration

The ResultID.Code can take the following values. For more information, refer to **ResultID Class**.

Value	Description
CONNECT_E_ADVISELIMIT	Advise limit exceeded for this object
CONNECT_E_NOCONNECTION	The client has no callback registered
DISP_E_TYPEMISMATCH	Type mismatch
E_BADRIGHTS	The item's access rights do not allow the operation
E_BADTYPE	The server cannot convert the data between the specified format and/or requested data type and the canonical data type
E_DEADBANDNOTSET	The item deadband has not been set for this item
E_DEADBANDNOTSUPPORTED	The item does not support deadband
E_DUPLICATENAME	Duplicate name not allowed
E_FAIL	Unknown error
E_INVALID_PID	The specified property ID is not valid for the item
E_INVALIDARG	An invalid parameter was passed to a method call
E_INVALIDCONFIGFILE	The server's configuration file is an invalid format
E_INVALIDCONTINUATIONPOINT	The continuation point is not valid
E_INVALIDFILTER	The filter string is not valid
E_INVALIDHANDLE	The handle value is not valid

E_INVALIDITEMID	The item ID does not conform to the server's syntax
E_NOBUFFERING	The server does not support buffering of data items that are collected at a faster rate than the group update rate
E_NOTFOUND	The requested object (e.g. a public group) was not found
E_NOTSUPPORTED	The server does not support writing of quality and/or timestamp
E_PUBLIC	The requested operation cannot be done on a public group
E_RANGE	The value is out of range
E_RATENOTSET	There is no sampling rate set for the specified item
E_UNKNOWNITEMID	The item ID was refused by the server
E_UNKNOWNPATH	The item's access path is not known to the server
RPC_S_CALL_FAILED	Remote procedure call failed
RPC_S_SERVER_UNAVAILABLE	The RPC server is currently not available
S_CLAMP	A value passed to write was accepted but the output was clamped
S_DATAQUEUEOVERFLOW	Not every detected change has been returned since the server's buffer reached its limit and had to purge the oldest data
S_INUSE	The operation cannot be performed because the object is being referenced
S_UNSUPPORTEDRATE	The server does not support the requested data rate but will use the closest available rate
WIN_S_FALSE	The function was partially successful
WIN_S_OK	Operation succeeded

Appendix 2 - QualityID Codes

The Quality.FullCode can take the following values. For more information, refer to **QualityID Class**.

Value	Description
OPC_QUALITY_BAD	Bad quality. Reason unknown.
OPC_QUALITY_COMM_FAILURE	Bad quality. Communications have failed and there is no last known value.
OPC_QUALITY_CONFIG_ERROR	Bad quality. There is as server configuration problem, such as the item in question has been deleted.
OPC_QUALITY_DEVICE_FAILURE	Bad quality. Device failure detected.
OPC_QUALITY_EGU_EXCEEDED	Uncertain quality. The returned value is outside the EGU limits defined for item.
OPC_QUALITY_GOOD	Good quality.
OPC_QUALITY_LAST_KNOWN	Bad quality. Communications have failed but there is a last known value available.
OPC_QUALITY_LAST_USABLE	Uncertain quality. A data source has not provided the server with a data update within the expected time period. The last known value is returned. Note, this is different from the OPC_QUALITY_LAST_KNOWN quality, which is used when the server is unable to read a value from a device. In this case, a data source has failed to write a value to the server in an unsolicited manner.
OPC_QUALITY_LOCAL_OVERRIDE	Good quality. The value has been overridden. This may indicate that an input has been disconnected and the returned value has been manually "forced".
OPC_QUALITY_NOT_CONNECTED	Bad quality. It has been determined that an input is disconnected, or that no value has been provided by data source yet.

OPC_QUALITY_OUT_OF_SERVICE	Bad quality. The item is off scan, locked, or inactive.
OPC_QUALITY_SENSOR_CAL	Uncertain quality. The value has either exceeded the sensor's limits (limit bits should be set to 1 or 2), or the sensor is known to be out of calibration (limit bits should be 0).
OPC_QUALITY_SENSOR_FAILURE	Bad quality. A sensor failure has been detected. Lth limit bits may provide additional information.
OPC_QUALITY_SUB_NORMAL	Uncertain quality. The value is derived from multiple sources, and fewer than the required number are good.
OPC_QUALITY_UNCERTAIN	Uncertain quality. No specific reason known.
OPC_QUALITY_WAITING_FOR_INITIAL_DATA	Bad quality. No value has been provided to the server yet.

Appendix 3 - QualityID LimitBits and Name

The full quality code is 16 bits: VVVVVVVVQQSSSSLL **where** V=vendor, Q=quality, S=substatus, L=limit.

Quality

QQ	Bit Value	Definition	Notes
0	00SSSSLL	Bad	The value is not userful for the reasons indicated by the substatus.
1	01SSSSLL	Uncertain	The quality of the value is uncertain for the reasons indicated by the substatus.
2	10SSSSLL	N/A	Not used by OPC.
3	11SSSSLL	Good	The quality of the value is Good.

Note: Servers that do not support quality information must return 3 (Good). It is also acceptable for a server to return Bad or Good (0x00 or 0xC0) and to always return 0 for substatus and limit.

Substatus for Bad Quality

SSSS	Bit Value	Definition	Notes
0	000000LL	Nonspecific	The value is bad but no specific reason is known.
1	000001LL	Configuration Error	There is a server-specific problem with the configuration (e.g., the item has been deleted from the configuration).
2	000010LL	Not Connected	The input that is required to be logically connected is missing. This quality may indicate that no value is available at this time for a reason such as the data source did not provide the value.
3	000011LL	Device Failure	A device failure has been detected.
4	000100LL	Sensor Failure	A sensor failure has been detected. The limit field may provide additional diagnostic information.
5	000101LL	Last Known Value	Communications have failed; however, the last known value is available. Note that the age of the value can be determined from the TIMESTAMP value in OPCITEMSTATE.
6	000110LL	Communications Failure	Communications have failed. There is no last known value available.
7	000111LL	Out of Service	The block is off-scan or otherwise locked. This quality is also used when the active state of the item or the group containing the item is InActive.
8		N/A	Not used by OPC.

Note: Servers that do not support substatus information should return 0.

Substatus for Uncertain Quality

SSSS	Bit Value	Definition	Notes
0	010000LL	Nonspecific	Indicates that there is no specific reason why the value is uncertain.
1	010001LL	Last Usable Value	Whatever was writing this value has stopped. The returned value should be regarded as "stale." Note that Last Usable Value is different from a bad value with substatus 5 (Last Known Value), which specifically indicates a detectable communications error on a "fetched" value. Last Usable Value indicates the failure of some external source to send a value within an acceptable period of time. The age of the value can be determined from the TIMESTAMP value in OPCITEMSTATE.
2-3		N/A	Not used by OPC.
4	010100LL	Sensor Not Accurate	Either the value has "pegged" at one of the sensor limits (in which case the limit field should be set to 1 or 2) or the sensor is otherwise known to be out of calibration as indicated by some form of internal diagnostics (in which case the limit field should be 0).
5	010101LL	Engineering Units Exceeded	The value returned is outside of the limits defined for that parameter. Note that in this case the limit field indicates which limit has been exceeded but that does NOT necessarily mean that the value cannot move farther out of range.
6	010110LL	Sub-normal	The value is derived from multiple sources and has less than the required number of good sources.
7-15		N/A	Not used by OPC.

Note: Servers that do not support substatus information should return 0.

Substatus for Good Quality

SSSS	Bit Value	Definition	Notes
0	110000LL	Nonspecific	The value is good and there are no special conditions.
1-5		N/A	Not used by OPC.
6	110110LL	Local Override	The value has been overridden. Typically this is because the input has been disconnected and a manually entered value has been "forced."

| 7-15 | N/A | Not used by OPC. |

Note: Servers that do not support substatus information should return 0.

Limit

LL	Bit Value	Definition	Notes
0	QQSSSS00	Not Limited	The value is free to move up or down.
1	QQSSSS01	Low Limited	The value has "pegged" at some lower limit.
2	QQSSSS10	High Limited	The value has "pegged" at some high limit.
3	QQSSSS11	Constant	The value is a constant and it cannot move.

Note: The limit value is valid regardless of the quality and substatus values. In some cases, such as Sensor Failure, the limit value can provide useful diagnostic information. Servers that do not support limit information should return 0.

Appendix

Appendix 1 ResultID Codes
Appendix 2 QualityID Codes
Appendix 3 QualityID LimitBits and Name

Index