

What's New in QuickOPC

Version 5.31

Contents

What's New in QuickOPC 5.31	6
User Interface.....	6
OPC XML-DA Support.....	8
Components Core	10
Documentation	12
Examples	12
Packaging	12
What's New in QuickOPC 5.30	13
Conformance.....	13
Technologies	14
Packaging	14
Deployment.....	14
User Interface.....	14
Examples	15
Removed	15
What's New in QuickOPC 5.23	16
Technologies	16
Components Core	16
Development Models.....	18
User Interface.....	18
Instrumentation	19
Diagnostics	19
What's New in QuickOPC 5.22	20
User Interface.....	20
OPC Unified Architecture.....	20
OPC Alarms and Events.....	20
Components Core	20
All Supported OPC Specifications	20
OPC Unified Architecture.....	21
OPC Data Access	23
OPC Alarms and Events.....	23
Development Models.....	23
All Supported OPC Specifications	23

OPC Unified Architecture.....	23
Security	24
OPC Unified Architecture.....	24
Diagnostics	24
All Supported OPC Specifications	24
OPC Unified Architecture.....	25
Instrumentation	25
OPC Unified Architecture.....	25
Performance.....	25
Examples	25
Tools	26
Documentation and Help.....	26
What's New in QuickOPC 5.21	27
Installation	27
Technology.....	27
Licensing.....	27
Development Tools Integration	28
Development Models.....	29
Components.....	30
Examples	38
Tools	39
Documentation	39
Packaging	40
Licensing.....	40
What's New in QuickOPC-Classic 5.20	41
Development Tools Integration	41
Packaging	42
Licensing.....	42
Development Models.....	42
Components.....	45
Examples	49
OPC Interoperability.....	53
Details	54
Browsing Dialogs.....	54
What's New in QuickOPC-Classic 5.12	63
Technology	63

Packaging	63
Components.....	63
OPC Interoperability.....	65
Documentation and Help.....	65
Examples	66
Internal Changes	66
Bug Fixes	66
What's New in QuickOPC-Classic 5.11	67
Packaging	67
Technology.....	67
OPC Interoperability.....	67
Documentation	68
Examples	68
Installation	68
What's New in QuickOPC.NET 5.10	69
OPC Interoperability.....	69
Components.....	69
Packaging	69
Related Products.....	70
What's new in QuickOPC-COM 5.10.....	71
OPC Interoperability.....	71
Components.....	71
Packaging	71
Related Products.....	71
What's New in QuickOPC.NET 5.04	73
Technology.....	73
Installation	73
Related Products.....	73
What's New in QuickOPC.NET 5.03	74
Technology.....	74
Installation	74
Related Products.....	74
What's New in QuickOPC.NET 5.02	75
OPC Interoperability.....	75
Technology.....	75
What's new in QuickOPC-COM 5.02.....	76

OPC Interoperability.....	76
Components.....	76
Installation	76
What's new in QuickOPC-COM 5.01	77
Technology	77
What's New in QuickOPC.NET 5.00	78
Technology	78
Components.....	78
Documentation	80
Packaging	80
Removed	80
What's new in QuickOPC-COM 5.00	82
Technology	82
Components.....	82
Tools and Instrumentation.....	83
Documentation	83
Packaging	84
Removed	84

What's New in QuickOPC 5.31

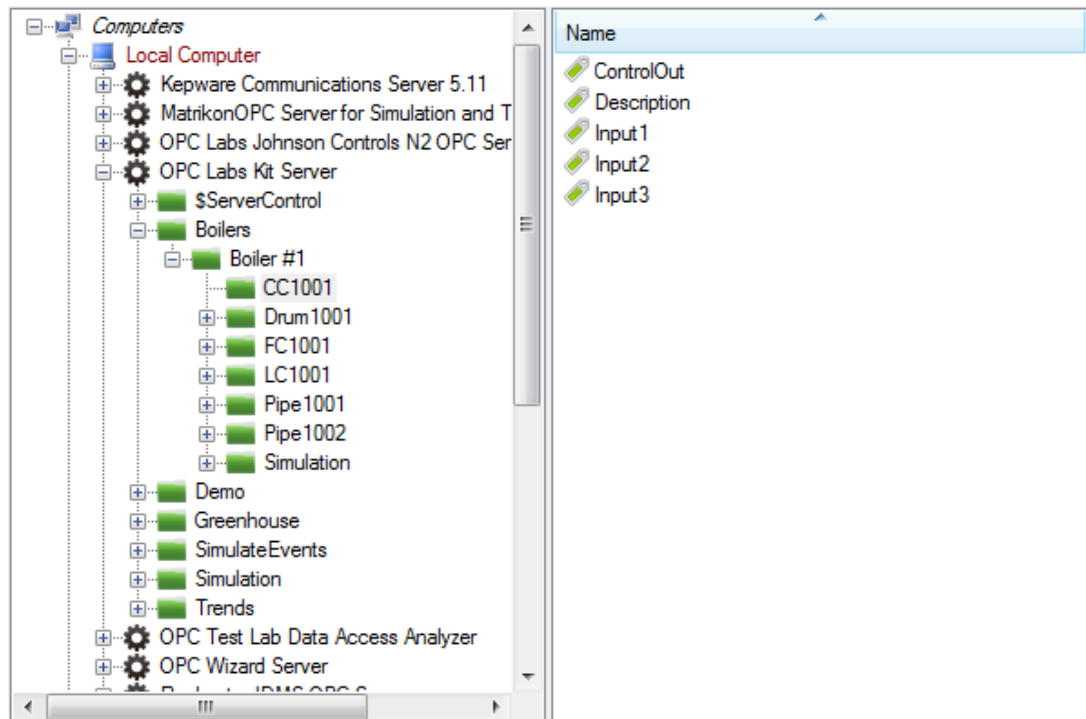
Key changes:

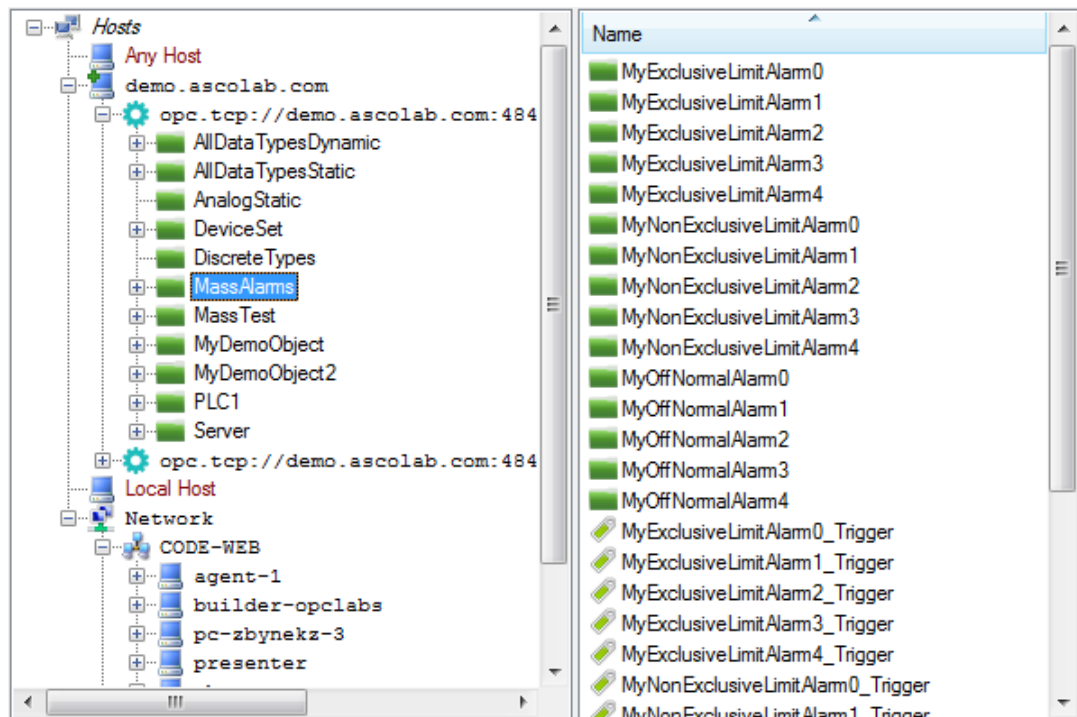
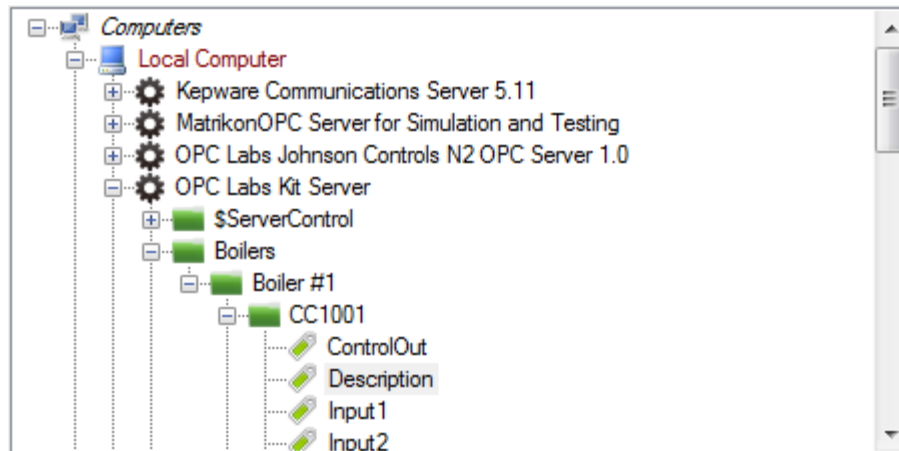
- *Improvements in user interface components*
- *Support for OPC XML-DA specification*

User Interface

- We have introduced two new components, **OpcBrowseControl** and **UABrowseControl**. These are browsing controls that can be placed onto your forms, configured to provide different kinds of OPC browsing, and cooperate with other controls on the form. Their functionality is similar to (parts of) **OpcBrowseDialog** and **UABrowseDialog**, respectively. Unlike with the dialogs, you have full freedom in creating your own visual appearance and behavior of the form, in case that the standard browsing dialog is not enough, or when you need closer integration with other parts of your application.

The controls can be configured to provide a tree view only, a list view only, or a combined tree view and list view. Examples of the controls in action are on the pictures below:





- The **OpcBrowseDialog** and **UABrowseDialog** dialog components (and the new **OpcBrowseControl** and **UABrowseControl** controls) now support multi-selection. It can be enabled by setting the **MultiSelect** property inside **OpcBrowseDialog.Mode** or **UABrowseDialog.Mode** to true.
- In relation to the multi-selection support described above, the relevant members of **OpcBrowseDialog.InputsOutputs** and **UABrowseDialog.InputsOutputs** have been moved

to a new **CurrentNodeDescriptor** property, so that the multi-selection data can be placed into the **SelectionDescriptors** collection alongside it. Analogically, the relevant members of **OpcBrowseDialog.Outputs** and **UABrowseDialog.Outputs** have been moved to a new **CurrentNodeElement** property, so that the multi-selection data can be placed into the **SelectionElements** collection alongside it.

- There are new ways to control some finer aspects of the **OpcBrowseDialog** and **UABrowseDialog**. For example, the **Mode.ShowListBranches** property (defaults to **true**) controls whether the branches of the tree are also displayed in the list view.
- The browsing dialogs now have splitters that allow the user to influence the sizes of various parts of the dialog.
- When you set the **Simulated** property of the **OpcBrowseDialog** or **UABrowseDialog** to true, the dialog will provide its contents from a pre-defined, simulated view of the world, with fake networks, computers, OPC servers, and their contents. This can be useful for experimentation and testing, either by the developer during the design (right in Visual Studio), or by the end-user (if you expose this functionality in your application), when the environment is not accessible.
- In browsing dialogs, the context menu (right-click) on the computer or host nodes now provides a command to make a PING test to the selected computer or host.
- In browsing dialogs, when entering names of new nodes manually (such as for computers that are not listed on the network, or for servers or endpoints that are not discoverable), a tooltip appears below the edit box, explaining the type of information that needs to be entered.
- The browsing dialogs now contain additional small buttons above the tree view. The buttons allow to expand all nodes below the current node recursively, and/or to make the current node expanded, but collapse all its sub-nodes.
- All browsing dialogs now offer a choice to cancel any long-running operation (such adding a large number of nodes to the selection set).
- All browsing dialogs now change the mouse cursor when some operation takes longer time.

OPC XML-DA Support

- QuickOPC.NET now directly supports OPC XML-DA Specification Version 1.01 (Released). There are no new public components for this – the XML-DA support is seamlessly

integrated with OPC Data Access. You can work with OPC XML-DA servers just as with COM-based OPC Data Access servers, simply by specifying the server's URL in place of the usual machine name and server class (ProgID). Accesses to OPC-DA and OPC XML-DA can be freely mixed, even in the same method call. This transparency has been achieved by several generalizations in the object model and API, described further below.

- The **ServerDescriptor** (which previously contained mainly the **MachineName** and **ServerClass** properties) has been generalized to be able to describe both COM and XML based servers. The primary information in the **ServerDescriptor** is now its **UrlString**. For OPC COM servers, the URL is composed in such a way that it contains the original **MachineName** (now named **Location**) and **ServerClass** properties. For OPC XML servers, their URL can be used as the **UrlString** directly. This design makes the **ServerDescriptor** backwards compatible, and developers that use COM servers only do not have to make any difficult changes.
- The **ServerDescriptor** now contains an additional **NetworkSecurity** property. This object can optionally specify the network credentials used when connecting to an OPC XML server.
- The **ServerElement** now has additional **UrlString** and **Location** properties.
- The **ServerDescriptor** and **ServerElement** have a new **Technology** property, indicating whether the server is an OPC COM or an OPC XML server.
- At some places (where both OPC COM and OPC XML are covered), the **MachineName** property has been renamed to **Location** (and can contain either the machine name or a host name, which are essentially the same thing, but a common term covering both was needed).
- The [Server] attribute (**ServerAttribute**) for Live Mapping now has a **UrlString** property (named argument), allowing you to specify an OPC XML-DA server for the mapping.
- The **ServerElementCollection** is no longer keyed by the server's **Guid** (because OPC XML-DA servers do not have a GUID), but instead it is keyed by a **ServerUrl** (for COM servers, the server URL contains the GUID in it).
- OPC XML-DA does not identify OPC items by a single string as OPC COM. Instead, it uses an additional string, and only the combination of the two identifies an item in an OPC XML-DA server. The original string that roughly corresponds to OPC item ID is called "ItemName" in the OPC XML-DA specification; in QuickOPC, we use the existing **ItemId** property for it. The additional string is called "ItemPath" in the OPC XML-DA specification; in QuickOPC, we

have introduced a new **NodePath** property for it. This property appears both in the **DANodeDescriptor** and **DANodeElement**. When you use the information received from the OPC XML browsing, both **ItemId** and **NodePath** are filled in the **DANodeElement**, and you can easily convert it to a **DANodeDescriptor** and it will “just work”. If you are getting the nodes (items) from elsewhere, you need to create a **DANodeDescriptor** that contains both these strings (at least in general case; some OPC XML-DA server do not use them both).

- The **DAItemIdTemplateAttribute.TemplateString** has been renamed to **ItemIdTemplate**, and a new property covering a template string for the OPC XML-DA “**ItemPath**” has been added, named **NodePathTemplateString**.
- OPC XML-DA identifies properties by a string (a XML qualified name), instead of the numerical code used in OPC COM. For this a reason, a **DAPropertyDescriptor** and a **DAPropertyElement** now have an additional **QualifiedName** property. When the component returns the **DAPropertyElement** from browsing, it fills the information it knows about. When you pass the **DAPropertyDescriptor** to the component, you can fill in either **PropertyId**, or **QualifiedName**, or both. The component will use whatever is available and at the same time usable by the underlying technology. For the most common case, standard (well-known) properties can always be identified using their numerical ID, and the component will look up their qualified name automatically.
- The **DAPropertyElement** now contains an additional **ItemPath** property. For properties coming from OPC XML servers, if the property can be accessed as an item as well, the property is filled in with the string that together with the **ItemId** identifies the OPC item that corresponds to the OPC property.
- The **DAPropertyDialog** now has a **PropertyDescriptor** property instead of a **PropertyId** property.

Components Core

- In .NET, the main methods constituting the functionality of **EasyDAClient**, **EasyAEClient**, and **EasyUAClient** components, have been extracted into **IEasyDAClient**, **IEasyAEClient**, and **IEasyUAClient** interfaces, respectively. The remaining methods and method overloads, which simply build upon the core interface methods, have been re-implemented as extension methods on the interface. Also, at many places, arguments and properties that used to accept the concrete **EasyXXClient** now accept the interface **IEasyXXClient** instead. In most languages (certainly in C# and Visual Basic), this refactoring does not cause any change in syntax of your code. The new design allows to supply a different implementation

of the component's interface where needed, and is currently used for simulation purposes in the browsing controls and dialogs.

- All parameter and policy objects (and their constituents) now consistently have a static **Default** property. This allows for a cleaner code where an arguments is needed, but you want to supply a default (there is now no need to call the default constructor).
- The **ServerDescriptor** has new properties, all related to the original **ServerClass** property, allowing to individually retrieve or modify parts of the server class designation. The **ProgId** property contains a ProgID of the server (and is an empty string if the ProgID is not given). The **Clsid** property contains the CLSID of the server in a form of a **Guid** (and is **Guid.Empty** if no CLSID for the server is given). The **ClsidString** property contains the CLSID string of the server (and is an empty string if no CLSID for the server is given). The **ObjectId** property contains the object ID, i.e. a ProgID, {CLSID}, or ProgID/{CLSID}.
- For ease of use, added **ServerDescriptor.FromServerElement**, and a corresponding implicit conversion operator.
- The overloads of **EasyUAClient.DiscoverServers** method that accept the "application types" argument have been renamed to **DiscoverApplications**, in order to better describe the actual functionality that they provide.
- The **EndpointUrlString** property of the **UAEndpointDescriptor** has been renamed to just **UrlString**.
- The **DAPPropertyDescriptor** no longer derives from the **DANodeInsteadDescriptor**. Instead, property descriptor appears together with a separate node descriptor argument where needed. **DAPPropertyArguments** now also have an additional **NodeDescriptor** property for this purpose.
- Renamed some **UABrowsePath** members: **BrowsePathElements** to **Elements**, **GetLastBrowsePathElement** to **GetLastElement**, **AppendBrowsePathElement** to **AppendElement**.
- Renamed the **IsolatedAdaptableParameters** property to just **IsolatedParameters**.
- Removed the **DefaultIsolatedAdaptableParameters** property. The isolated (adaptable) parameters are now always initialized to a constant default.
- Removed certain obsolete members.

Documentation

- The main documentation chapters are now primarily structured by the development models, and not by the OPC specifications.

Examples

- Added **EasyOpcNetDemoXml**: This is a source of the Demo application for OPC “Classic” (OPC XML-enabled) that ships with the QuickOPC.NET product. The application shows most product functions, including the browsing forms, OPC property access, and event-based subscriptions. The defaults are pre-filled for OPC XML-DA demo server, but the application is written in such a way that it can handle COM servers as well.
- Added **Listview1** example in VB.NET: Shows how (Windows Forms) ListView items can be populated with OPC data, either using explicit Read, or with a subscription.
- Added **SubscribeMultipleItems** example in C++.

Packaging

- Some types have moved from the **EasyOpcClassic** assembly to the **EasyOpcClassicInternal** assembly. As a result, more projects will also need to reference the **EasyOpcClassicInternal** assembly.
- New assemblies (do not have to be explicitly referenced, but need to be deployed together with your applications): **OpcLabs.EasyOpc**, **OpcLabs.EasyOpcClassicNative**, **OpcLabs.EasyOpcClassicNetApi**, **OpcNetApi**, **OpcNetApi.Com**, **OpcNetApi.Xml**.

What's New in QuickOPC 5.30

Key changes:

- *OPC Certification*
- *.NET Framework 4.5*

Conformance

- QuickOPC is officially certified by OPC Foundation for "UA Generic Client" profile (UA CTT 1.02.0.164), Serial Number 1312CS0045,
<http://www.opcfoundation.org/Default.aspx/certifiedproduct.asp?MID=Compliance&certificate=1312CS0045>,
<http://www.opcfoundation.org/Default.aspx/CertifiedProducts.asp?MID=Compliance> .



- The default settings of QuickOPC are set for best interoperability. If you require best OPC compliance, do one of the following:
 - a) Set the static property **EasyUAClient.AdaptableParameters** to **EasyUAClient.AdaptableParameters.OpcCompliance**. This will influence all newly created **EasyUAClient** instances that have their **Isolated** property set to **false** (the default).

- b) On a newly created **EasyUAClient** instance, set the **Isolated** property to **true**, and set the **IsolatedAdaptableParameters** to **EasyUAdaptableParameters.OpcCompliance**. This will influence this particular **EasyUAClient** instance only.

Technologies

- .NET Framework 4.5 is now the primary/minimal target for QuickOPC.NET development.
- Visual Studio 2012/2013 are now the targeted development tools with QuickOPC.NET.
- .NET code access security now uses the Level 2 transparency model.
- COM: Unchanged – any development tool capable of COM development can be used. For Microsoft tools, this should mean Visual C++ 6.0 or later, although we do not test for it specifically, and Visual Studio 2012/2013 is used for verification.

Packaging

- Separate Microsoft.Contracts assembly is no longer needed.
- Separate assemblies for functionality available only with .NET 4 are no longer needed.

Deployment

- Different Visual C++ redistributables (Visual C++ Redistributable Packages for Visual Studio 2013) are now needed.
- COM: Visual C++ redistributables are no longer needed for QuickOPC-COM deployments.

User Interface

- In QuickOPC-UA, by default, user notifications (such as certificate validation error, or domain name mismatches) are now shown in one common dialog, allowing multiple notifications be processed at the same time. The old behavior (each notification is shown its own dialog) is still available. The controlling parameter is **UAClientEngineParameters.UseParallelNotifier**.
- Various browsing dialogs have been enhanced by splitters, i.e. the user can now resize parts of the dialogs by moving the splitters as needed.

- The properties of **OpcBrowseDialog** (for OPC “Classic”) have been grouped into **Mode**, **Inputs**, **InputsOutputs**, and **Outputs** objects.
- The properties of **UABrowseDialog** (for OPC Unified Architecture) have been grouped into **Mode**, **Inputs**, **InputsOutputs**, and **Outputs** objects.

Examples

- Visual Studio examples are now mainly provided for Visual Studio 2012. They can be automatically converted to Visual Studio 2013.
- Earlier separate examples for .NET 4 were merged into the regular example projects/solutions.
- Examples for reactive programming model have been separated into a dedicated solution, because they need an installation of Microsoft Reactive Extension.

Removed

- COM: The bonus OPCItemGen application has been removed.
- COM: The “Portable” examples have been removed.

What's New in QuickOPC 5.23

Key changes: Support for Microsoft StreamInsight

Technologies

- QuickOPC now allows development for Microsoft StreamInsight. This support is provided through (separately licensed) StreamInsight Option for QuickOPC. The option is physically installed together with QuickOPC.NET and QuickOPC-UA by default. Shortcuts that relate to StreamInsight Option have their separate group under the product folder in the Start menu. To learn more, read the documents and study the examples under the "StreamInsight Option" group.

Components Core

- Various default "hold" and reconnection periods have been shortened, usually by a factor of approx. 5. This change applies to both QuickOPC Classic and QuickOPC-UA.
- The component's parameters (for all **EasyXXClient** objects) are now consistently reorganized into three groups: **SharedParameters** (static parameters that are always shared among object instances), **InstanceParameters** (parameters that can always be set independently for each object instance), and **AdaptableParameters** (parameters that are normally shared, but can be made specific to an instance if the **Isolated** property is set to 'true').
- The **EasyUAA adaptableParameters** class contains two static properties with pre-defined sets of parameters. These properties are called **Interoperability** and **OpcCompliance**. By assigning one of the two to the actual parameters of the component, the developer can quickly choose between the settings that provide the best interoperability (the default), or best compliance with OPC standards. Fine-tweaking of the individual settings is still possible.
- Additional overloads of the **DiscoverServers** allow you to specify e.g. the application types you are interested in, using the combination of flags from the **UAApplicationTypes** enumeration. This can be useful e.g. if you want to do hierarchical discovery and return also the discovery servers (which are not returned by default).
- The expanded node ID strings (e.g. in the **UANodeId** object) can now contain the namespace index (in place of, or in addition to, the namespace URI). Note that specifying the

namespace index alone, without the namespace URI, should only be used when absolutely necessary, because the namespace indices are not generally guaranteed to stay the same between sessions, although some servers may behave as such.

- The **UANodeId** object now has a string **StandardName** property. When the node ID represents one of the nodes defined by the OPC-UA standard, the **StandardName** property reflects the symbolic name of such node (e.g. "TypesFolder"). Conversely, when you set the **StandardName** property to a symbolic name, the node ID objects change its other properties to represent the node given by the standard name. For nodes that are not defined in the OPC-UA standard have, the **StandardName** property contains an empty string.
- The **UAClientSessionParameters** has a new property, **SessionConnectTimeout**, which allows to specify how long the connection attempt can take.
- In order to allow StreamInsight serialization, some property members in generic types that are based on their corresponding non-generic types have been renamed. The changes are described by the following table:

Type	Old member name	New member name
DAItemValueArguments<T>	Value	TypedValue
DAItemVtqArguments<T>	Vtq	TypedVtq
DAVtq<T>	Value	TypedValue
DAVtqResult<T>	Vtq	TypedVtq
EasyDAItemChangedEventArgs<T>	Vtq	TypedVtq
EasyUAMonitoredItemChangedEventsArgs<T>	AttributeData	TypedAttributeData
UAAttributeData<T>	Value	TypedValue
UAAttributeDataResult<T>	AttributeData	TypedAttributeData

- The **UAApplicationType** enumeration has been renamed to **UAApplicationTypes**, and its internal values have changed. The enumeration is now bit-coded, allowing for better capture and detection of various combinations of OPC-UA application types.
- The **CertificateAcceptancePolicy** has been moved from **UAClientEngineParameters** to **UAClientSessionParameters**. As the session parameters can be made isolated and

configured separately for each instance of the client component, you can now configure the certificate acceptance policy differently for each instance of the component.

Development Models

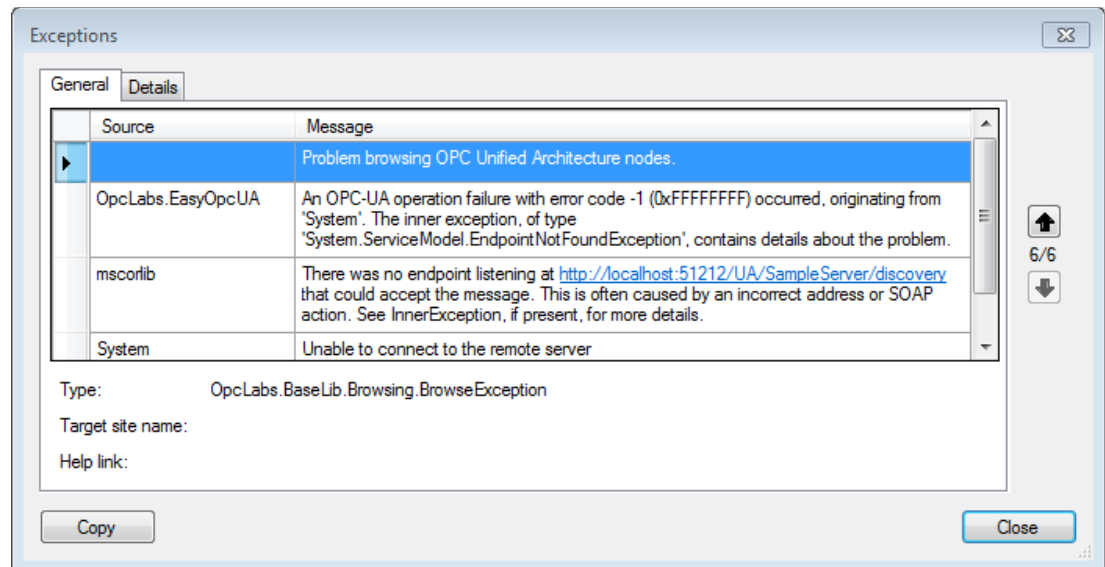
- Live Binding: When the Binding Errors dialog is shown in Visual Studio, it now has an additional “Details...” button. This button allows to view all details of the binding errors, using the same exception dialog box as mentioned with UI browsing dialogs.
- Live Mapping: Some classes have been renamed. The following table summarizes the changes.

Old type name	New type name
DAItemMapping	DAClientItemMapping
DAItemSource	DAClientItemSource
DAMapper	DAClientMapper
DAPropertyMapping	DAClientPropertyMapping
DAPropertySource	DAClientPropertySource
UADataMapping	UAClientDataMapping
UADataMappingSource	UAClientDataMappingSource
UAMapper	UAClientMapper

- Reactive Programming Model: In all reactive objects, replaced the public **Client** object (property) by a **ClientSelector** property. The client selector just holds the parameters the client object should have. The client object itself is not serialized. This approach allows full and easy serialization of reactive objects.

User Interface

- When an error occurs in dialogs for UI browsing, a small “...” button next to the error box allows detailed viewing of the exception(s). The exception box looks similarly to this:



The exception box has following features:

- One or more exceptions can be displayed.
- Each exception is displayed together with its inner exceptions that caused it.
- The Details tab provides technical information about the exception objects.
- The Copy button stores all available information in textual form into the clipboard, suitable for error reporting by the user.

Instrumentation

- The **EasyUAClientConfiguration** component now has a **LogEntry** event, which functions as a “projection” of the static **EasyUAClient.LogEntry** event. You can easily hook an event handler to this event to process log entries generated by the **EasyUAClient** component.

Diagnostics

- The **Diagnostics** and **DiagnosticsSummary** properties are now also available and filled in on the **EasyUAMonitoredItemChangedEventArgs** object, providing diagnostics information related to monitored items.

What's New in QuickOPC 5.22

Key changes: Improvements in OPC Unified Architecture

User Interface

OPC Unified Architecture

- A new node, labeled "Any", appears in **UABrowseDialog**, **UADDataDialog** and **UAHostAndEndpointDialog**, under the "Hosts" node. The user can manually add any server endpoint under this node (by typing its endpoint URL), even for hosts that are not exposed through discovery.
- The **UADDataDialog** now supports a different mode, activated by setting the **UserPickEndpoint** property to **true**. In this mode, the user is not limited to nodes from a pre-set OPC-UA server; instead, the user starts with choosing the host and server endpoint. In The **EndPointDescriptor** property in this mode contains the chosen endpoint, or (in case of multi-select) there is a new **EndpointDescriptors** property, with endpoint information for each selected node.

OPC Alarms and Events

- The **OpcBrowseDialog** component now supports two new values for its **SelectElementType** property: **OpcElementType.SourceCondition** and **OpcElementType.SourceSubcondition**. These settings allow browsing for conditions and subconditions associated with an event source node.

Components Core

All Supported OPC Specifications

- The **Isolated** flag of **EasyDAClient.ClientMode**, **EasyAEClient.ClientMode** and **EasyUAClient.ClientMode** becomes **EasyDAClient.Isolated**, **EasyAEClient.Isolated** and **EasyUAClient.Isolated**.
- The **LicensingException** class has been removed, and Microsoft's **System.ComponentModel.LicenseException** is used instead.

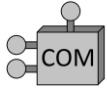
OPC Unified Architecture

- The OPC Unified Architecture components are based on version 1.02 of the .NET Stack from OPC Foundation.
- The component now attempts to detect whether a debugger is attached to the application, and if so, it uses a different, much slower keep-alive interval (this keep-alive interval is controlled by the **UASessionParameters.KeepAliveIntervalDebug** property, and defaults to 1 day). This allows the developer to break into the debugger and examine the status of the program, without causing the OPC-UA sessions be disconnected due to the fact that keep-alives are not being exchanged while the execution is suspended.
- The **AttributeType (AttributeTypeCode)** property of **UAWriteArguments** and **UAWriteValueArguments** has been removed. The component now determines the type based on the **AttributeId**. This works for all attributes except the **Value** attribute. When writing to a **Value** attribute, the type is given by a new **ValueType** (or **ValueTypeCode**) property.
- Similarly, the **AttributeType (AttributeTypeCode)** property in **UADataBinding** has been made obsolete. The new developments should use **ValueType (ValueTypeCode)** property instead, and it is only needed when the **Value** attribute is being bound.
- When writing to the **Value** attribute, unless the value type is given explicitly in the code, QuickOPC-UA now reads the **DataType** and **ValueRank** attributes of the given node, and determines the value type from the OPC-UA server. This gives more convenience, as the value type does not have to be specified in wider range of cases, but the developer should be aware of possible negative performance implications.
- Added overloads of **EasyUAClient.Write** and **WriteValue** methods that accept the attribute Id argument.
- The **WriteMultiple** and **WriteMultipleValues** method of **EasyUAClient** now return **UAWriteResult** instead of **OperationResult**. The **UAWriteResult** contains additional information in case of Write success: the **Clamped** flag and the **CompletesAsynchronously** flag.
- The new **IncludeSubtypes** flag in **UABrowseParameters** determines whether subtypes of the specified reference type should be returned by the browsing.
- A new **LocaleId** property on the **EasyUAClient.SessionParameters** allows the developer to specify the Locale id to be used by the OPC-UA server for localized strings.

- You can now call a static method **EasyUAClient.CloseAll** to close all unused sessions that are open to OPC-UA servers.
- Added a new overload of the **EasyUAClient.DiscoverServers** method that takes an input array or URL strings to attempt discovery on, and a flag indicating whether the discovery should be made in parallel.
- The component now internally subscribes to and monitors the **State** value in the **ServerStatus** of the OPC-UA server. If the server indicates that it is being shut down, the component disconnects from the server. A reconnection attempt will be made after a period configurable by the **EasyUAClient.SessionParameters.ServerShutdownRetrialPeriod** property.
- The **UAObjectIds** class now contains much more objects – in fact, it contains all objects defined by the OPC-UA Specification. Some objects have been renamed in order to keep 100% alignment with the standard OPC-UA Information Model, e.g. **Objects** is now **ObjectsFolder**, **Root** is now **RootFolder**, etc. The old names have been preserved but made obsolete.
- Completed parsing support for extended OPC-UA browsing paths (with the use of [] notation, and namespaces specified by their URI; both for references and for targets).
- Added a '+' operator to **BrowsePath** and **UABrowsePath**. Consequently, a '+=' (in C#) is available as well. This allows easy appending of browse name (or browse element) to the browse path.
- There is now an implicit conversion from **UAQualifiedName** to **UABrowsePathElement**, resulting in shorter code for creation of OPC-UA browse paths in some cases.
- There is now an implicit conversion from an integer (**Int32**) to a **UAIndexRange**, therefore an index range containing just a single element can be written simply as the element index, without explicitly constructing the **UAIndexRange** object.
- One-dimensional index range lists can now be easily constructed using the **UAIndexRangeList.OneDimension** static method, passing it either the single index, or minimum and maximum indices.
- Added **EasyUAClient.HostParameters** and **DiscoveryParameters** properties. Moved parts (mainly discovery-related) of **EasyUAEngine.EngineParameters** to these new objects. Also, added corresponding **EasyUAClient.IsolatedHostParameters** and **IsolatedDiscoveryParameters** properties.

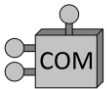
- The **SessionHoldPeriod** of **EasyUAClient.EngineParameters** becomes a **HoldPeriod** of **EasyUAClient.SessionParameters**.

OPC Data Access



- By checking the “Non-variant arrays” box in EasyOPC Options Utility -> OPC-DA Globals -> Engine Parameters, the developer indicates that the hosting application does not want all arrays provide by OPC-DA servers be converted to arrays of VARIANTS.

OPC Alarms and Events



- A new method on **EasyAEClient** object, **QuerySourceConditions**, allows to obtain the event conditions associated with the givens source. This new method is available both in COM and .NET components.
- By checking the “Non-variant arrays” box in EasyOPC Options Utility -> OPC-A&E Globals -> Engine Parameters, the developer indicates that the hosting application does not want all arrays provide by OPC-A&E servers be converted to arrays of VARIANTS.

Development Models

All Supported OPC Specifications

- Live Mapping: Added support for mapping tags (using the **MappingTag** attribute). Mapped members can be tagged with one or more tags. When performing operations, the developer can specify that only members with certain tag, or combination of tags, will be affected.
- Live Mapping: Added support for deferred mapping (the mapped object does not have to exist at the time of mapping, but instead a deferred mapping function provides it dynamically, when it is referenced).

OPC Unified Architecture

- Added OPC-UA Modelling (preliminary). This development model automatically provides .NET objects that logically correspond to OPC-UA information model. The operative parts of this model are internally implemented using Live Mapping. Currently, only certain standard OPC-UA variable types are supported; custom types are not yet available. Supported concrete classes include: **UAPropertyNode**, **UABaseDataVariableNode**; for UA Data Access: **UADataItemNode**, **UAAanalogItemNode**, **UATwoStateDiscreteNode**, **UAMultiStateDiscreteNode**.

Security

OPC Unified Architecture

- Added property **UACertificateAcceptancePolicy.AllowUserAcceptCertificate** (defaults to **true**). When the component is run in user-interactive mode, the user can be prompted to accept a server certificate that have otherwise failed the certificate validation process.
- Added static methods to **UAUserIdentity** object for easy creation of various user tokens: **CreateAnonymousIdentity**, **CreateKerberosIdentity**, **CreateUserNameIdentity**, and **CreateX509Certificate**.
- The subject name of the application instance certificate that is used or created by the component can now be set using the **EasyUAClient.EngineParameters.ApplicationCertificateSubject** property. Other properties of the certificate, as well as client description used when opening a session, can be changed using **EasyUAClient.EngineParameters.ApplicationName**, **ApplicationUriString**, and **ProductUriString** properties.
- The new **AllowedMessageSecurityModes** property of the **UAEndpointSelectionPolicy** gives you a possibility to completely enable or disable certain classes of endpoints from selection, based on their security mode. You can combine the value of this property from **SecurityNone**, **SecuritySign**, and **SecuritySignAndEncrypt** flags. The default value allows all types of endpoints be selected.
- The **PreferMessageSecurity** property of the **UAEndpointSelectionPolicy** has been renamed to **MessageSecurityPreference**, and it now has three possible values: **Negative** (to prefer non-secure endpoints), **None** (to follow server-provided endpoint security level), and **Positive** (to prefer secure endpoints).

Diagnostics

All Supported OPC Specifications

- The **OperationResult** object (and therefore all result objects derived from it as well) now has a **Diagnostics** property, which contains a collection of warning and informational diagnostics entries gathered, performing the requested operation. Currently, the diagnostics information is only filled in for OPC-UA operations.
- The additional **OperationResult.DiagnosticsSummary** string contains a textual summary of diagnostics information, one message per line.

OPC Unified Architecture

- A new **DiagnosticsMasks** property on the **EasyUAClient.SessionParameters** allows the developer to specify the types of vendor-specific diagnostics to be returned by the OPC-UA server in the responses. The information from the server responses is then made available in the Diagnostics collection of **OperationResult** objects.
- **EasyUAClient** now detects stale timestamps.
- **EasyUAClient** now detects missing timestamps.
- **EasyUAClient** now checks the server signature.
- **EasyUAClient** now checks the server nonce.
- **EasyUAClient** now thoroughly checks the outcomes of the **CreateSession**.
- **EasyUAClient** now performs many other checks required by OPC compliance test.

Instrumentation

OPC Unified Architecture

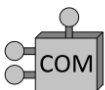
- The **EasyUAClient** component now has a static **LogEntry** event. This event is raised for loggable entries originating in the OPC-UA client engine and the **EasyUAClient** component. Each event notification carries a **LogEntryEventArgs** object, which contains information such as the source of the event, the event message, event type. Timestamp, etc. The component does not store log the events on itself, but the developer can receive the notifications and implement any kind of logging with them.

Performance

- Build 397.1: Significantly improved speed of **EasyDAClient.GetPropertyValues** and **GetMultiplePropertyValues** methods.

Examples

- The OPC-UA Demo Application now uses the **UAHostAndEndpointDialog**, making it possible for the user to also browse for servers on hosts other than the local host.
- Added new VBA examples in Excel (using QuickOPC-COM): **ReadAndWriteValue.xls**, **ReadAndDisplayMultipleValues.xls**, **SubscribeToMultipleItems.xls**.



Tools

- Test Tools for .NET have been added to the installation. There are two separate testing tools: One for OPC “Classic”, and one for OPC Unified Architecture. The tools allow exploration of various API objects and members, testing of the UI dialogs, live data view with multiple items, etc. The test tools are not installed by default; you need to select “Custom Install”, and then specifically enable them on the “Select Components” page in the installation wizards. After installation, the test tools can be reached through the Start menu, in the “Test Tools” group under product menu.

Documentation and Help

- Added a document describing the Test Tools.

What's New in QuickOPC 5.21

Key changes:

- *QuickOPC-Classic Installation Merged with QuickOPC-UA*
- *Introduced Browsing Dialogs for OPC-UA*
- *Live Binding and Live Mapping for OPC-UA*

Installation

- The QuickOPC-Classic (.NET and COM) and QuickOPC-UA are now merged into one installation package.
- OPC Core Components Redistributable updated to version 3.00.105.1 (resolves some OPC server enumeration problems on x64 platform).
- The installation package file is now code-signed, increasing security, improving trust, and providing better download experience from the Web. We use COMODO Code Signing CA 2 certificate.

Technology

- Built with Visual Studio 2012 Update 1.
- Due to OPC UA SDK requirements, the minimum platform requirement for QuickOPC-UA is now .NET Framework 3.5 Service Pack 1 "Full". For QuickOPC.NET, the "Client Profile" is still sufficient.

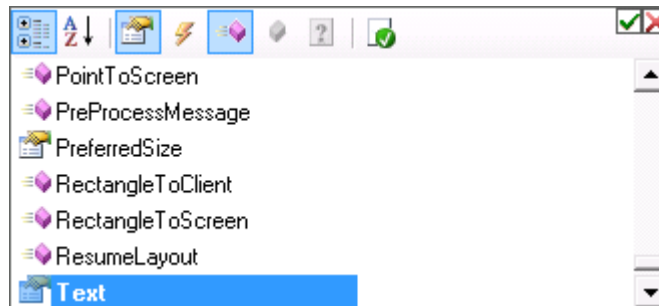
Licensing

- A single license can now cover all products or subset of them (QuickOPC-COM, QuickOPC.NET, QuickOPC-UA).
- Added a **LicenseInfo** property to **EasyDAClient**, **EasyAEClient**, **EasyUAClient** components. This property allows the developer to retrieve or view all parameters of the installed and recognized license (useful for troubleshooting licensing issues).
- A viewer form is provided for **LicenseInfo** in the designer.

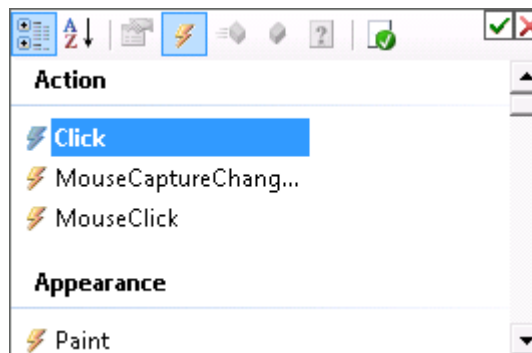
Development Tools Integration

- QuickOPC-UA assemblies are now also using Microsoft Code Contracts, their contract assemblies are shipped with the product, and the contracts are visible in the reference documentation.
- QuickOPC-UA components are automatically installed into the Visual Studio toolbox.
- Improved member name selection in Visual Studio (applies when you are selecting a target member for value binding, or when selecting a source event). The new member name editor shows different icons for different member kinds, annotates the members with number of overloads if needed, shows member information (such as type, and description) in the tooltip, allows filtering (e.g. Properties, Event, Methods, Fields), alphabetical or categorized view, and reverting to default member.

Here is how it may look like for selecting target members:



Here is how it may look like for selecting source events:




- In Visual Studio, wherever a property represents a time span (in milliseconds), an editor is provided that allows entering the value comfortably:

<input checked="" type="checkbox"/> Enabled	Reset to 0	
<input type="text" value="0"/> days	<input type="text" value="0"/> h	<input type="text" value="0"/> min
<input type="text" value="1"/> s	<input type="text" value="0"/> ms	
<input type="text" value="0.00001157407"/>	days	
<input type="text" value="0.0002777778"/>	hours (h)	
<input type="text" value="0.01666667"/>	minutes (min)	
<input type="text" value="1"/>	seconds (s)	
<input type="text" value="1000"/>	milliseconds (ms)	

Development Models

- The Live Binding model, previously available for OPC-Classic only, is now also supported for OPC-UA.

With Live Binding, no manual coding is necessary to obtain OPC connectivity. You simply use the Visual Studio Designer to configure bindings between properties of visual or non-visual components, and OPC data. All functionality for OPC reads, writes, subscriptions etc. is provided by the QuickOPC components.

The live binding model is allowed by a combination of the existing **BindingExtender** component, and a new **UABinder** component - .

- The Live Mapping model, previously available for OPC-Classic only, is now also supported for OPC-UA.

The Live Mapping model allows you to write objects that correspond logically to a functionality provided by whatever is “behind” the OPC data. For example, if part of your application works with a boiler, it is natural for you to create a boiler object in the code. You then describe how your objects and their members correspond to OPC data – this is done using attributes to annotate your objects. Using Live Mapping methods, you then map your objects to OPC data, and perform OPC operations on them.

When you subscribe to or read from OPC attributes, incoming OPC value changes can directly set corresponding properties in your objects, without any further coding. You can then focus on writing the code that handles these property changes, abstracting away the fact how they come in.



To give a concrete example, here is a piece of code of a .NET object annotated for live mapping with OPC Unified Architecture, in C#:

```
[UAType]
class BoilerInputPipe
{
    [UANode]
    public FlowTransmitter FlowTransmitter1 = new FlowTransmitter();

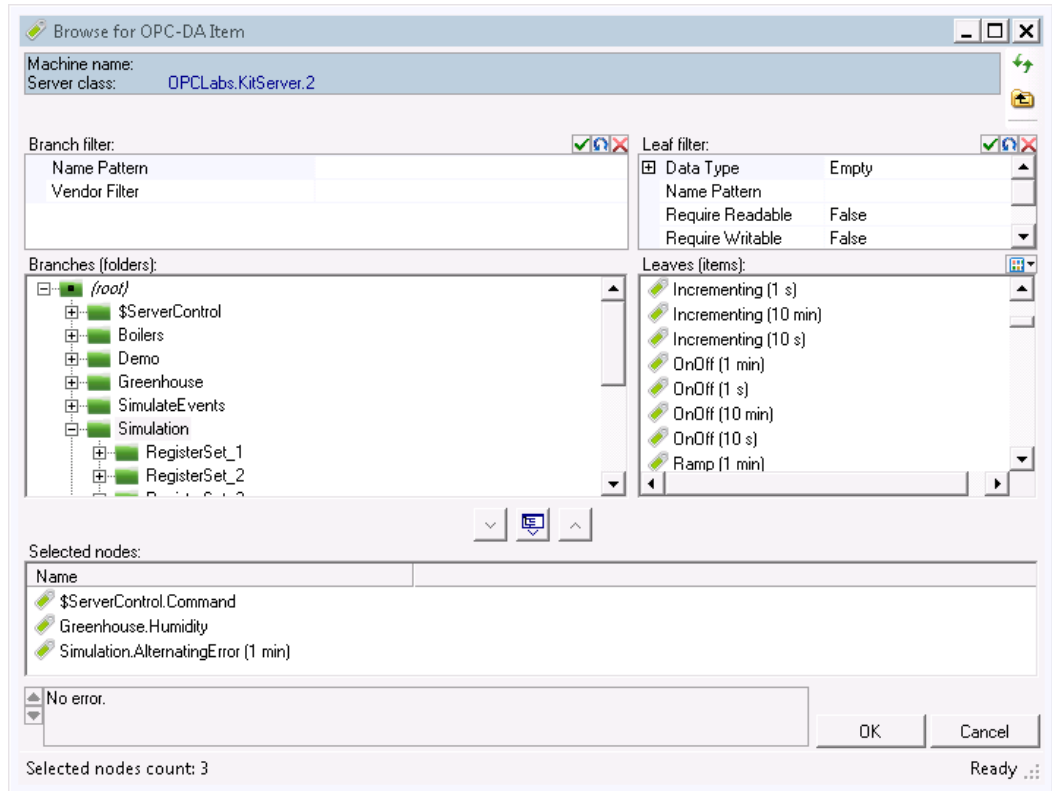
    [UANode]
    public Valve Valve = new Valve();
}
```


- The Reactive Programming model, previously available for OPC Data Access only, is now also supported for OPC Unified Architecture data. The **UAMonitoredItemChangedObservable** class serves as push-based notification provider for changes in OPC-UA monitored items. The **UAWriteValueObserver** is a push-based notification receiver that writes incoming values into an attribute of an OPC-UA node.
- The Reactive Programming model, previously available for OPC Data Access only, is now also supported for OPC Alarms&Events. The **AENotificationObservable** class serves as push-based notification provider for OPC-A&E notifications. The **AEAcknowledgeConditionObserver** is a push-based notification receiver that acknowledges OPC-A&E conditions.


Components


- New: **EasyDAClientConfiguration** component-  allows you to use Visual Studio designer to configure the static properties of **EasyDAClient** object, such as the security parameters, reconnect and topic retrieval delays, etc.
- New: **EasyAEClientConfiguration** component-  allows you to use Visual Studio designer to configure the static properties of **EasyAEClient** object, such as the security parameters, reconnect delays, buffer size, etc.
- The **DAItemDialog** component now retains the filter setting for each node between the invocations of the dialog, making it faster for the user to navigate during the subsequent invocations.

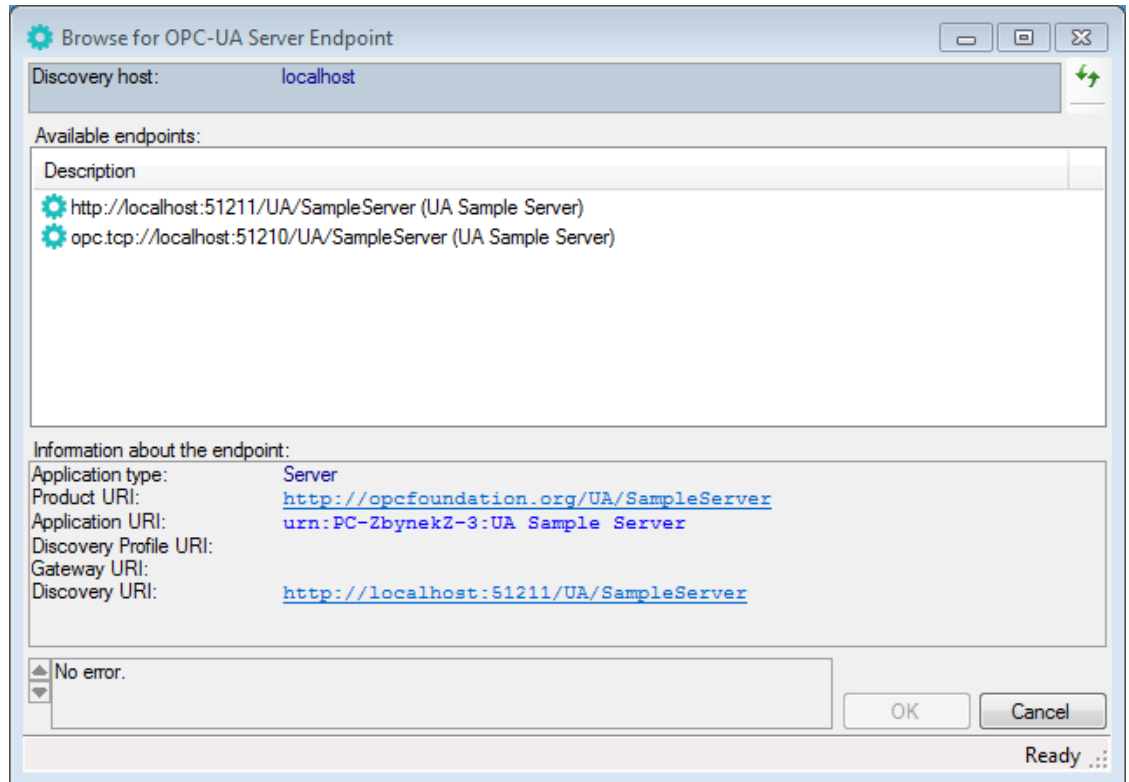
- Using the **MultiSelect** property, the **DAItemDialog** can now be switched to a multi-selection mode, allowing the user to add multiples nodes to the selection set, or remove them; the selected set is carried over to next invocation as well:




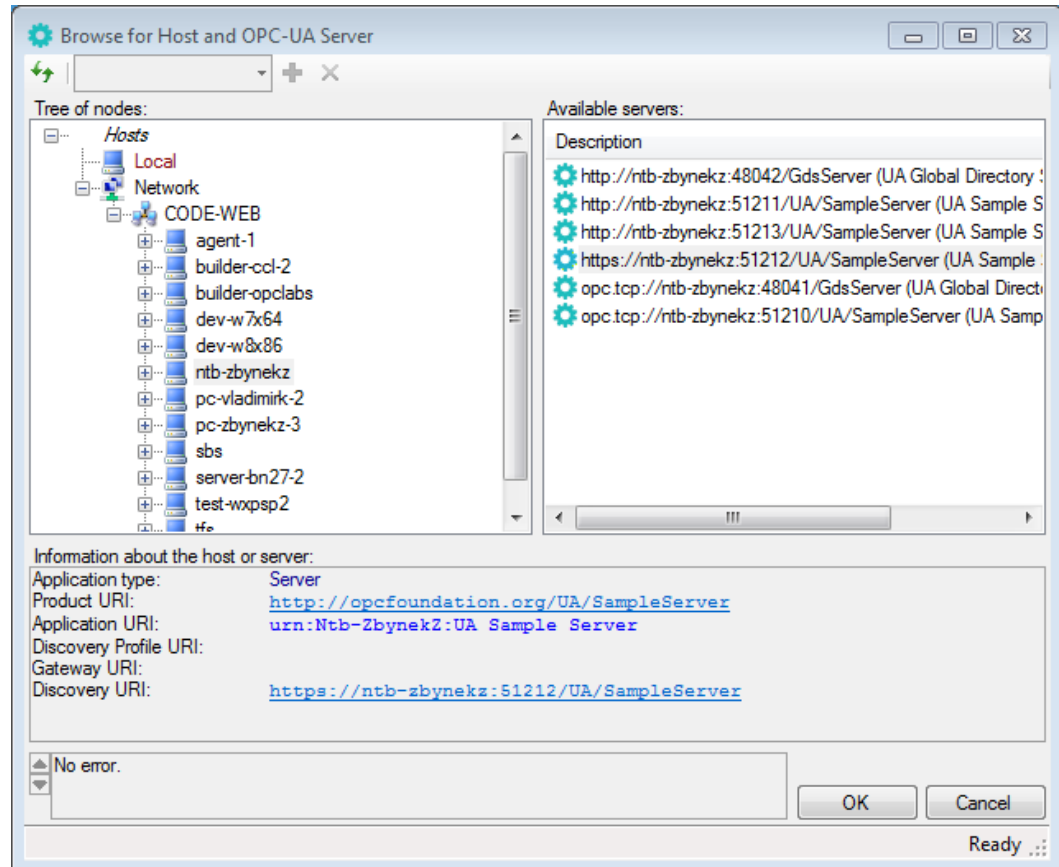
- The **OpcBrowseDialog** component has been extended by an ability to browse OPC-DA access paths, using a new an additional element type (**DAAccessPath**).
- The **EasyUAClient** component -  is now automatically installed into the Visual Studio Toolbox.
- QuickOPC-UA is now based on OPC UA SDK 1.01 .NET Stack 333.0 Stable.
- QuickOPC-UA now allows specifying the user identity to be used by the client, either by anonymous token, user name and password, X.509 certificate, or Kerberos (issued) token. The **UAClientSessionParameters** object now has a **UserIdentity** property with necessary information for this.

- It is now possible to call (invoke) methods inside OPC-UA servers, and pass input and output arguments to/from them, using **EasyUAClient.CallMethod** or **EasyUAClient.CallMultipleMethods**.
- It is now possible to browse for methods in OPC-UA servers, using one of **EasyUAClient.BrowseMethods** overloads. The method nodes can also be included in more generic **EasyUAClient.BrowseNodes** browsing, with the inclusion of **UABrowseParameters.Methods** in the parameters.
- An **EasyUAClient.BrowseVariables** method now allows browsing for OPC-UA variables, i.e. data variables or properties at one. In order to line up better with the proper OPC-UA terminology, the original **BrowseVariables** method has been renamed to **BrowseDataVariables**.
- For OPC-UA, the class for exceptions arising from OPC operations has been renamed from **OpException** to **UAException**, in order to prevent clash with the **OpException** class used in OPC "Classic".
- When performing a local discovery of OPC-UA servers, the component now attempts to connect to multiple possible discovery endpoints in parallel, speeding up the discovery. This behavior is controlled by the **ParallelDiscovery** property in the **UAClientEngineParameters** object.
- New: **EasyUAClientConfiguration** component-  allows you to use Visual Studio designer to configure the static properties of **EasyUAClient** object, such as the user identity, certificate handling rules, endpoint selection policy, etc.

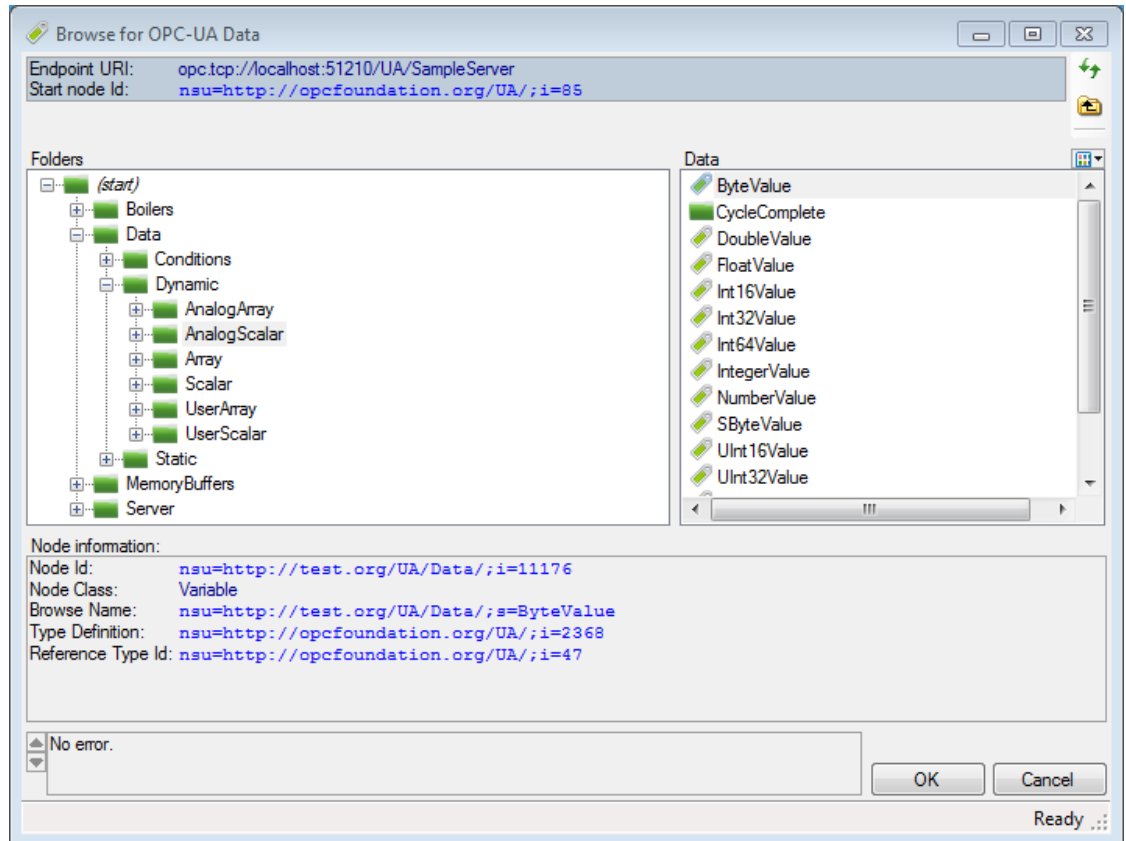
- Introduced a browsing dialog for OPC-UA server endpoints,  - a **UAEndpointDialog** component:




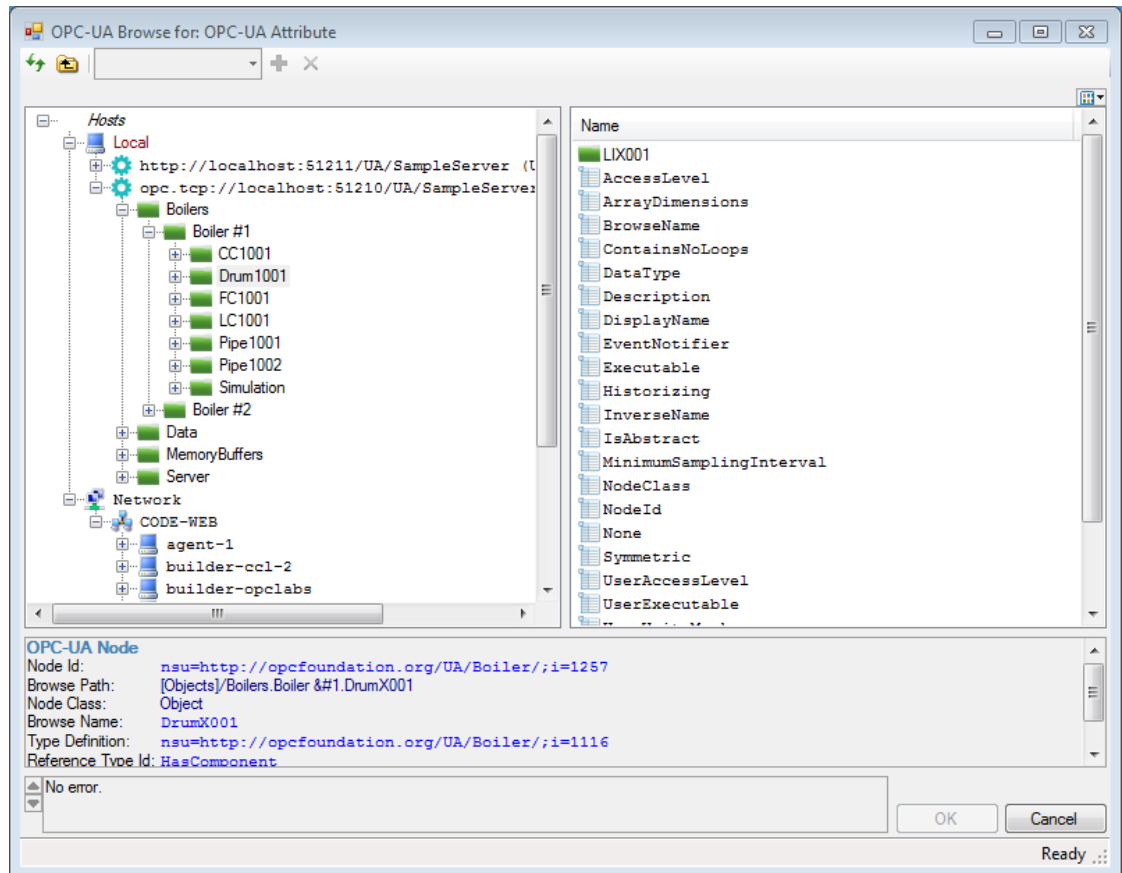
- Introduced **UAHostAndEndpointDialog** -  component: A browsing dialog for choosing a host (computer) together with an endpoint of an OPC-UA server residing on it.



- Introduced a browsing dialog for choosing an OPC-UA data node, a **UaDataDialog** component:



- Introduced a generic OPC-UA browsing dialog with various OPC-UA elements from which the user can select. This dialog, a **UABrowseDialog** component - , can be configured to serve many different purposes.



- A new object, **UANodeDescriptor**, is now used at most places where **UANodeId** has been used previously. The **UANodeDescriptor** contains the **NodeId** as a property. The reason for this change is to allow a parallel placement of browse paths into node descriptors. There are conversions and conversion operators (including implicit conversions) between these classes, so most code can remain without change.
- Individual or combined parts of **UANodeId** can now be separately retrieved or modified by their corresponding properties, such as **ExpandedText**, **Identifier**, **NamespaceUriString**, **NodeIdType**, etc.

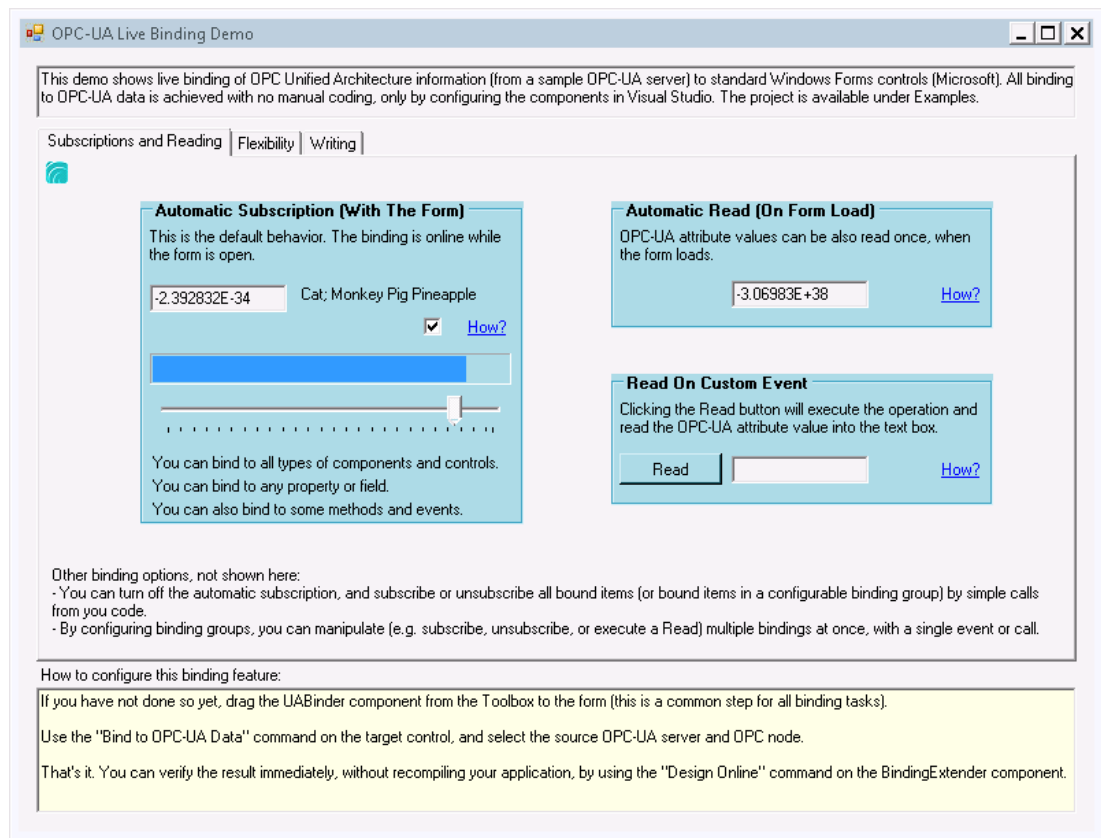
- Individual or combined parts of **UAQualifiedName** can now be separately retrieved or modified by their corresponding properties, such as **ExpandedText**, **Name**, and **NamespaceUriString**.
- Instead of **MaximumAge** property, previously available directly on the **UReadArguments** object, you now need to go one level deeper, and use **MaximumAge** property of **UReadArguments.ReadParameters** instead.
- There is now a new **ClientMode** member on the **EasyUAClient** object. Its **Isolated** property, when set to **true**, allows the connections made by using this particular instance of **EasyUAClient** be separate from others.
- Added new (in some cases, just documented existence of) static classes with standard OPC-UA Node IDs: **UDataTypeIds**, **UAMethodIds**, **UAObjectIds**, **UAObjectTypelds**, **UReferenceTypelds**, **UVariableIds**, and **UVariableTypelds**.
- Extension method **UAApplicationType.IsServer** combines recognition of **Server** and **ClientAndServer** application types.
- Extension methods on **UANodeElement** allow finer categorization of nodes returned by browsing, e.g. **IsDataVariable** and **IsProperty** test whether a given node is a Data Variable or a Property.
- Added possibility to serialize and deserialize practically all QuickOPC-UA objects (and their collections and dictionaries) using **Serializable** attribute and/or **ISerializable** interface. This serialization type is typically used with **BinaryFormatter** for storing objects in a binary format.
- Added possibility to serialize and deserialize practically all all QuickOPC-UA objects (and their collections and dictionaries) using **XmlSerializer** and **IXmlSerializable**. This serialization provides objects storage in XML format.
- New properties **UseCustomSecurity** and **TurnOffCallSecurity** in **EasyMachineParameters** (for connections to OPCEnum) and in **EasyDASClientParameters** and **EasyAEClientParameters** objects (for connections to OPC-DA and OPC-A&E servers) allow finer settings of COM security parameters.
- The **DefaultInstance** property (in various objects) has been renamed to **SharedInstance**.

- It is now possible to assign own parameter objects to the static and instance properties of **EasyAEClient**, **EasyDAClient** and **EasyUAClient** components. Previously, you could change properties of these parameters objects, but you could not assign a whole new parameter object to them.

Examples

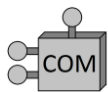
- Examples for QuickOPC-UA have been downgraded from Visual Studio 2010 to Visual Studio 2008, making it possible for users of Visual Studio 2008 to work with them easily.
- Many examples in C# have been made available in VB.NET as well. This includes the **DocExamples** project (for OPC "Classic") which contains examples that appear in the reference documentation.
- QuickOPC-UA examples have been merged into common solutions with QuickOPC.NET Examples.
- The **EasyOPC-UA Demo Application** now makes use of the browsing dialog for OPC-UA endpoints and data nodes.

- Added **UALiveBindingDemo** example: Shows live binding of OPC Unified Architecture information (from sample OPC server) to standard Windows Forms controls (Microsoft). All binding to OPC-UA data is achieved with no manual coding, only by configuring the components in Visual Studio.



- Added simple **ConsoleApplication1** example in Visual F#.

Tools



- The EasyOPC Options Utility now allows configuring specific COM security parameters for OPCEnum and OPC server connections.

Documentation

- The conceptual documentation for all QuickOPC products (QuickOPC-COM, QuickOPC.NET and QuickOPC-UA) is now merged into one document.

- The reference documentation for .NET products (QuickOPC.NET and QuickOPC-UA) is now merged into one package.
- The reference documentation for QuickOPC.NET now contains VB.NET examples, alongside the (pre-existing) C# examples.



Packaging

- For OPC-UA, there are new assemblies, **EasyOpcUAExtensions**, **EasyOpcUAForms**, and **EasyOpcUAInternal**. In your OPC-UA projects, it is now almost always necessary to explicitly reference the **EasyOpcUAInternal** assembly, as well as the **BaseLib** and **BaseLibExtensions** and assemblies.
- Various types have been moved from **EasyOpcClassic** assembly to the **EasyOpcClassicInternal** assembly. For OPC “Classic” usage, it is now almost always necessary to explicitly reference the **EasyOpcClassicInternal** assembly, as well as the **BaseLib** and **BaseLibExtensions** and assemblies.

Licensing

- Also QuickOPC-UA licenses now contain information about software version numbers covered, and a software release date covered, for easier maintenance coverage.

What's New in QuickOPC-Classic 5.20

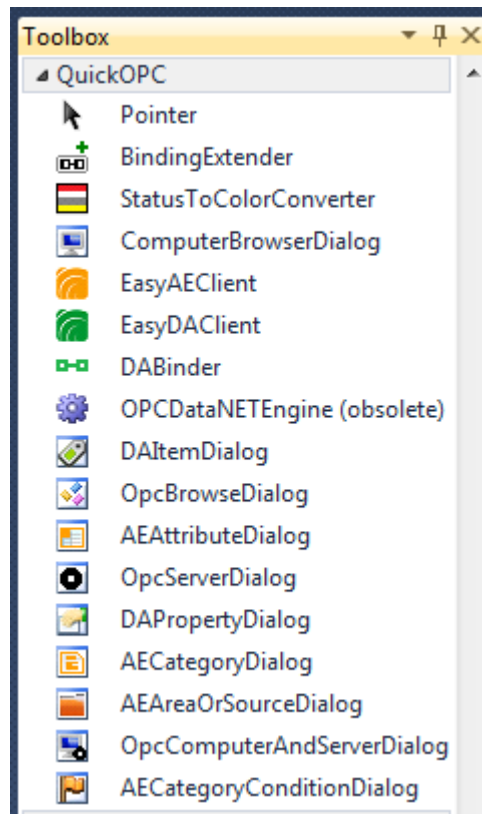
Key changes: OPC Live Mapping and Binding, New Browsing Dialogs

Note: Most changes in this version affect QuickOPC.NET only.



Development Tools Integration

- The components are now automatically installed into Visual Studio Toolbox:



- API members are now annotated using ReSharper (<http://www.jetbrains.com/resharper/>) attributes, such as the most common **[CanBeNull]** and **[NotNull]** attributes. ReSharper users benefit from the annotations, as possible improper usages are recognized and highlighted by ReSharper's code inspection feature.
- The API now uses Microsoft Code Contracts (<http://msdn.microsoft.com/en-us/devlabs/dd491992.aspx>, <http://research.microsoft.com/en-us/projects/contracts/>) to

express coding assumptions. This results in higher consistency in error checking, and the contracts are also explicitly visible in the reference documentation, showing the developer clearly what are the expectations about method inputs and outputs, etc. Developers who want to gain further benefit can enable Code Contracts in their projects, and use the QuickOPC contract assemblies supplied with the product for achieving better code quality.



Packaging

- There are new assemblies, and some existing assembly names have changed.
- The assemblies are now installed into the GAC (Global Assembly Cache) by the setup program. You do not have to install assemblies into the GAC on the runtime computers, though.

Licensing

- The licenses now contain information about software version numbers covered, and a software release date covered. This means that licenses issued can already take into account the maintenance terms and period, and do not have to be-issued for software covered by the maintenance.



Development Models

- Added a new development model: Live Binding.

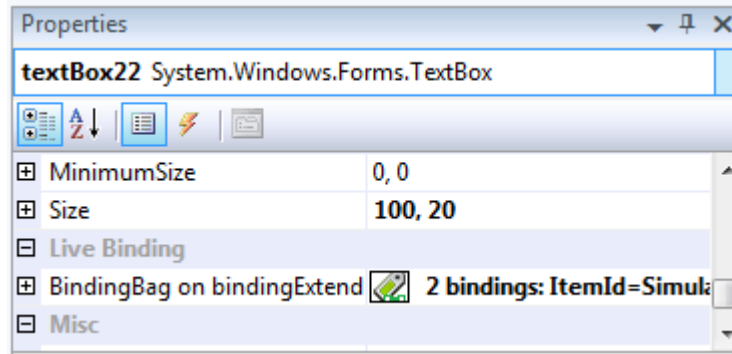
With Live Binding, no manual coding is necessary to obtain OPC connectivity. You simply use the Visual Studio Designer to configure bindings between properties of visual or non-visual components, and OPC data. All functionality for OPC reads, writes, subscriptions etc. is provided by the QuickOPC components.

The live binding model is allowed by the new **BindingExtender** and **DABinder** components that you drag from the toolbox to the designer surface. You can then use extender commands in the Properties window, or on the context menu of each component (control), to bind the component's properties to OPC data:

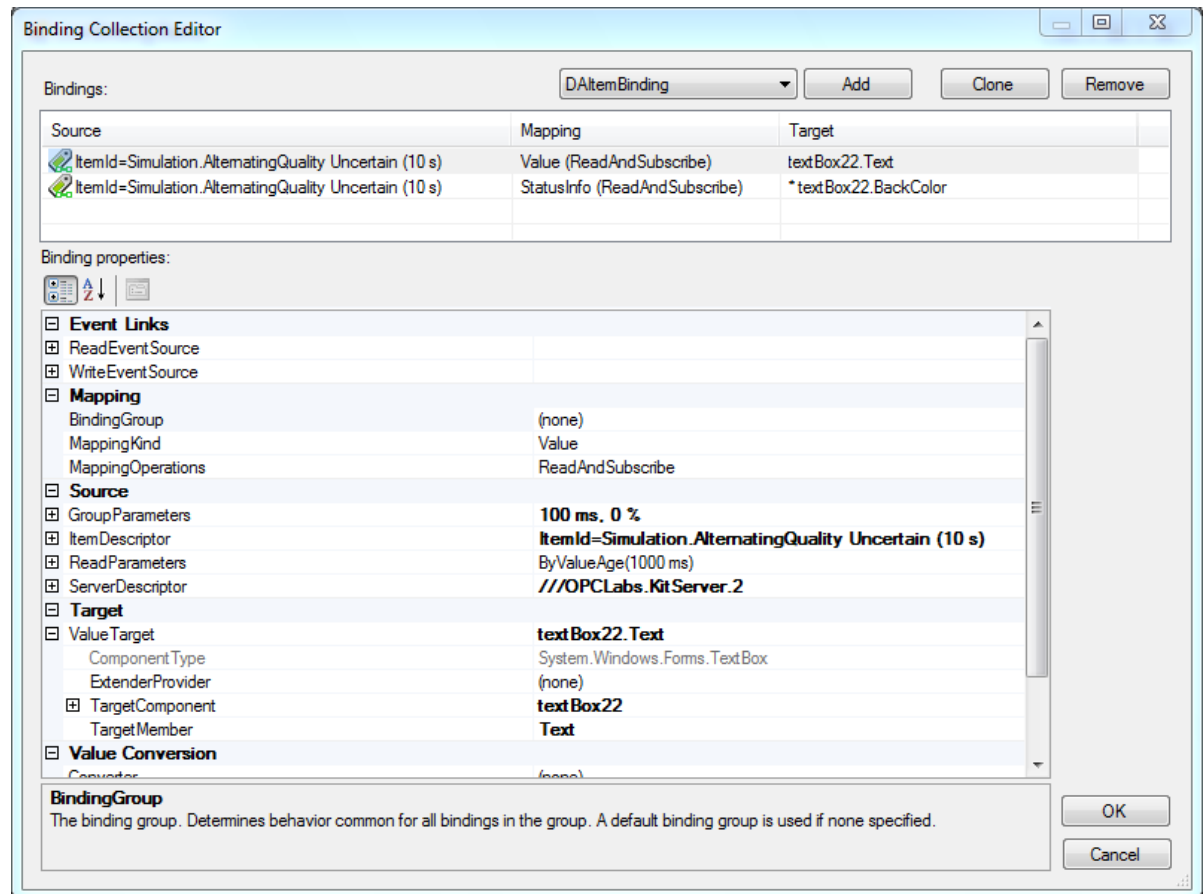


[Bind to OPC Item...](#), [Bind to OPC Property...](#),
[Edit Live Bindings...](#), [Remove Live Bindings](#)

After selecting the “Bind to OPC Item” command, you select the OPC data you want to bind to, and you are done. The currently configured bindings appear as additional entries in the Properties window:



You can also invoke the full-featured binding editor, and configure all aspects on bindings – either on individual component (control), or for a whole container (form) at once:



The Live Binding model is currently available for Windows Forms (or UI-less) containers only.

- Added a new development model: Live Mapping.

The Live Mapping model allows you to write objects that correspond logically to a functionality provided by whatever is “behind” the OPC data. For example, if part of your application works with a boiler, it is natural for you to create a boiler object in the code. You then describe how your objects and their members correspond to OPC data – this is done using attributes to annotate your objects. Using Live Mapping methods, you then map your objects to OPC data, and perform OPC operations on them.

For example, when you subscribe to OPC items, incoming OPC item changes can directly set corresponding properties in your objects, without any further coding. You can then focus on

writing the code that handles these property changes, abstracting away the fact how they come in.

To give an example, here is a piece of code of a .NET object annotated for live mapping with OPC Data Access, in C#:

```
[DAType]
class BoilerInputPipe
{
    [DANode]
    public FlowTransmitter FlowTransmitter1 = new FlowTransmitter();

    [DANode]
    public Valve Valve = new Valve();

    [DANode, DAItem]
    public bool InAlarm { get; set; };
}
```

- Added a new development model: Reactive Programming.

This development model merges the world of Microsoft Reactive Extensions (Rx) for .NET with OPC. The Reactive Extensions (<http://msdn.microsoft.com/en-us/data/gg577609.aspx>) is a library to compose asynchronous and event-based programs using observable collections (data streams) and LINQ-style query operators.

- The new development models, along with the original and traditional Procedural Coding Model, can be freely mixed in the same project.

Components

- Both QuickOPC-COM and QuickOPC.NET: Reduced latency of OPC reads and writes.



- Added support for .NET generics. This allows achieving type safety with OPC data values which are of variant type. There are now new generic types that accept type parameters determining the type of the value; for example, the developer can use **DAVtq<int>** for OPC data that carry integer values. There are also new generic methods that accept or return the generics.

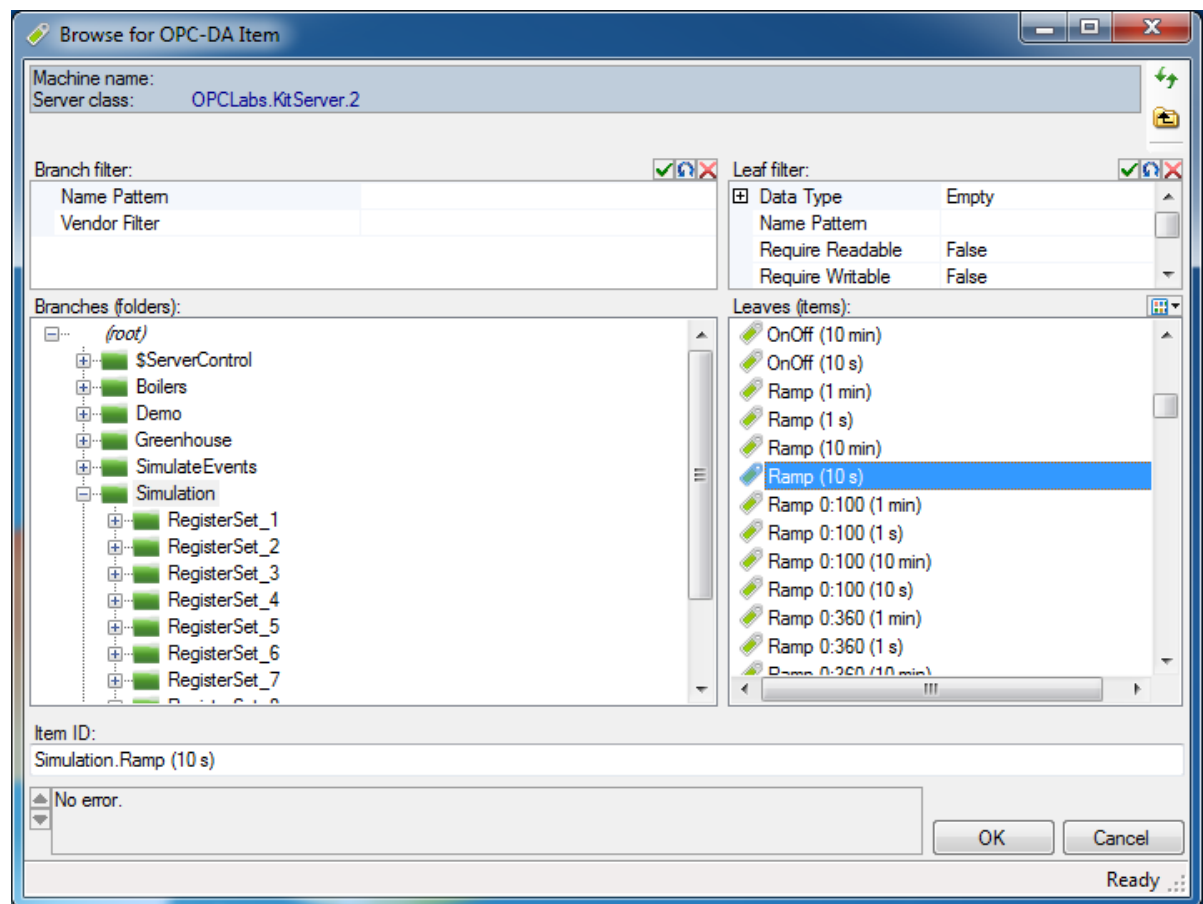
The developers are encouraged to use generics, but the existing types remain unchanged and can be used as before.

- There are new namespaces, and some existing types (mainly the Windows Forms dialogs) have been moved to different namespaces. The new namespace structure consistently distinguishes

between different OPC specifications, and whether the type is specific to a technology (such as Windows Forms).

- **New browsing dialogs.** Existing dialogs for browsing OPC Data Access servers, items and properties have been greatly improved visually and in functionality. Completely new dialogs have been added, covering all necessary OPC Alarms&Events browsing. A “universal” dialog that can be configured to browse just about anything related to OPC is also provided.

In total, 10 different dialogs are available; [click here for details](#). Below is an example of one of the dialogs:



- The existing OPC browsing dialogs have been replaced by new ones, meaning that their names and namespaces have changed.
- Added support for specifying OPC items (or any nodes in address space) using browse paths (a series of “short” item names), instead of item IDs, for all operations (not only for browsing, as in

Version 5.12). The **DAItemDescriptor** now inherits from **DANodeDescriptor** and therefore contains a **BrowsePath** property that can be used for this purpose. Also, many methods that accepted just node ID have been changed to accept **DANodeDescriptor**, allowing you to choose between node ID or a browse path.

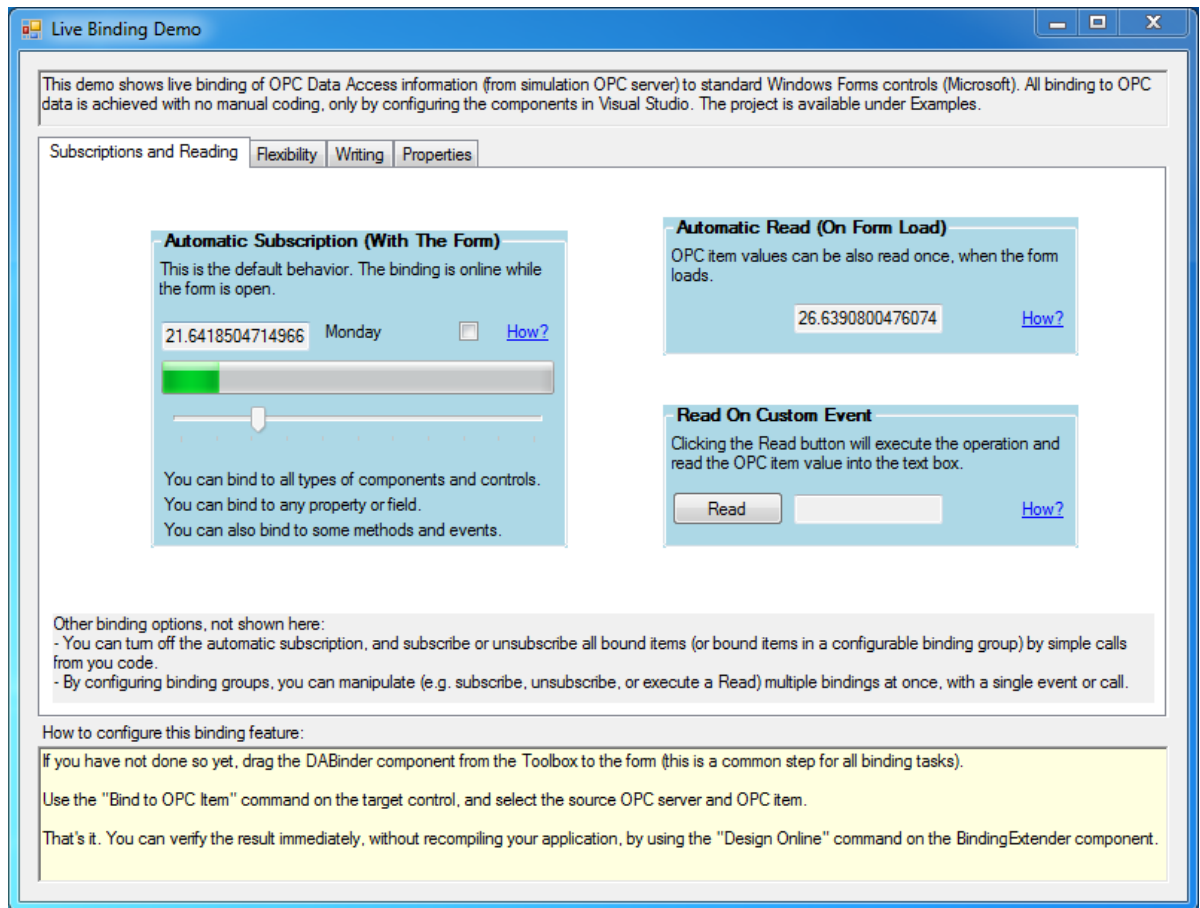
- **ChangeMultipleSubscriptions** and **UnsubscribeMultipleItems** methods now return “void”, as they always succeed unless there is a coding error.
- **UnsubscribeMultipleItems** method can be passed **IEnumerable<int>** in place of an array of handles.
- **DataSource** and **ValueAge** have moved from **DAClientMode** to a new object, **DAReadParameters**. There are now overloads of the **ReadXXXX** methods that accept a **valueAge** argument, or a argument of type **DAReadParameters**. This means that you can now easily specify the data source (cache or device), or desired value age, with each call.
- The **DAGroupParameters** object now has a **FromInt32** static method, and a corresponding implicit conversion operator, that allow it be constructed from an integer that represents the requested update rate (in milliseconds). This means that in C# and many other languages, you can simply use an integer update rate at all places where **DAGroupParameters** object is expected, if you are fine with specifying just the update rate and keeping the other properties of **DAGroupParameters** at their defaults.
- In browsing, it is easier to distinguish or select the proper type of server, with the new **ServerFamilies** enumeration, and the **Families** property on the **ServerCategories** object. As opposed to server category, the server family corresponds to e.g. OPC Data Access in general, without distinguishing the individual specification versions, such as OPC-DA 1.0, 2.0 or 3.0.
- New **DAPropertyId.GetName** and **GetPropertyType** methods can be used to obtain the string identifier of the property, or its type.
- Added the most generic OPC Alarms&Events address space browsing method, **BrowseNodes**. It combines the functionality of **BrowseAreas** and **BrowseSources**, and it also allows the widest range of filtering options by passing in an argument of type **AENodeFilter**.
- Added new method, **EasyAEClient.QueryCategoryConditions**, for finding out event conditions supported by given event category. Added new method, **EasyAEClient.QueryCategoryAttributes**, for finding out event attributes that the server can provide as part of an event notification within a given event category.

- New extension methods on **EasyAEClient** for OPC Alarms&Events: **FindEventCategory** and **FindEventCondition** return information about the given category given its ID, or about the given condition given its name.
- The **VarTypeUtilities** static class provides a **FromType** method that allows you to determine the COM-based type **VarType** (used in OPC operations) that corresponds to a given .NET **Type**.
- The **DAUtilities** static class provides methods that allow you to determine whether a given data type (**VarType**), percent deadband, update rate or value age are valid values in OPC Data Access.
- The **AEUtilities** static class provides methods that allow you to determine whether given data type (**VarType**), event severity, event type filter, or notification rate are valid values in OPC Alarms&Events.

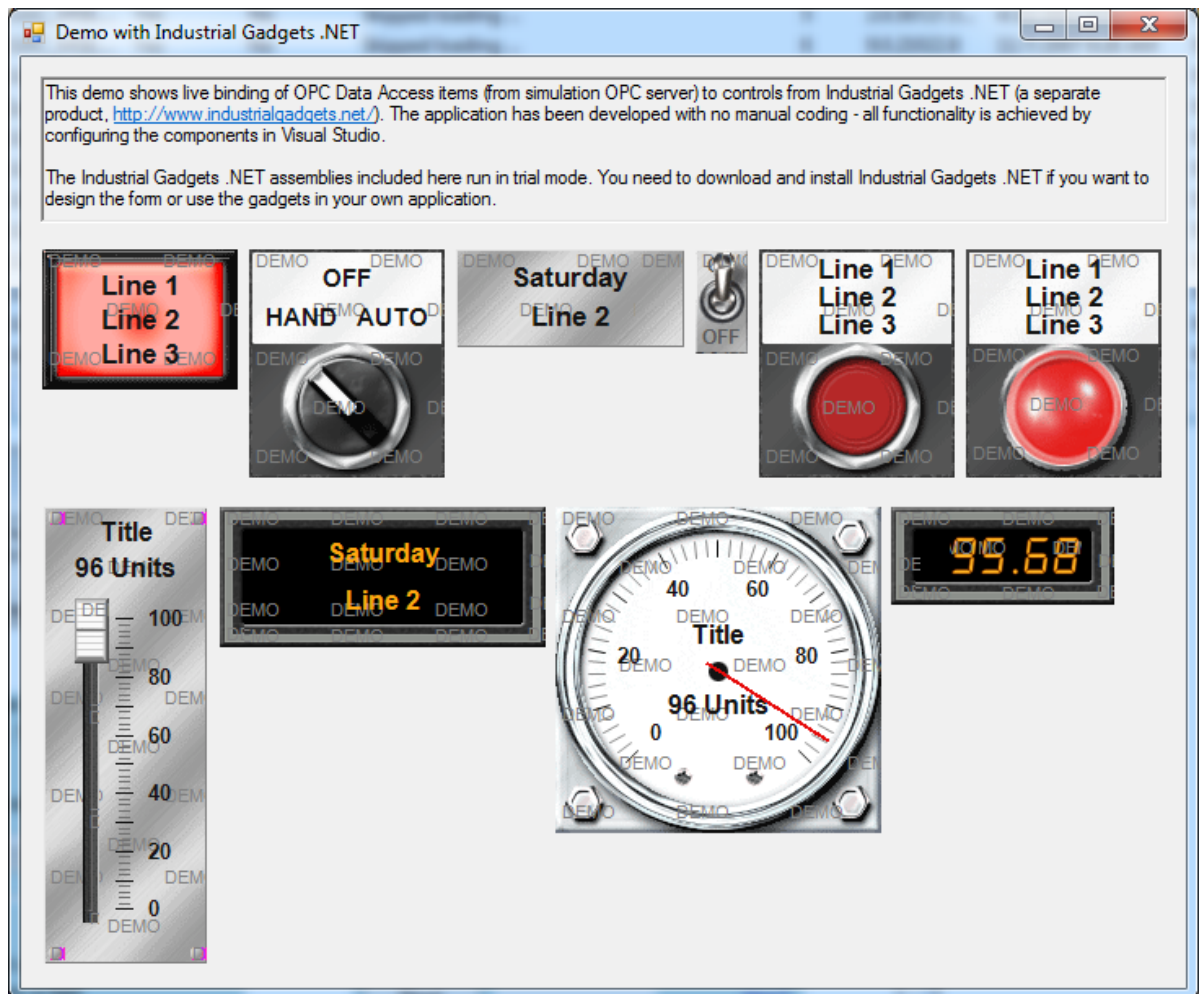
Examples



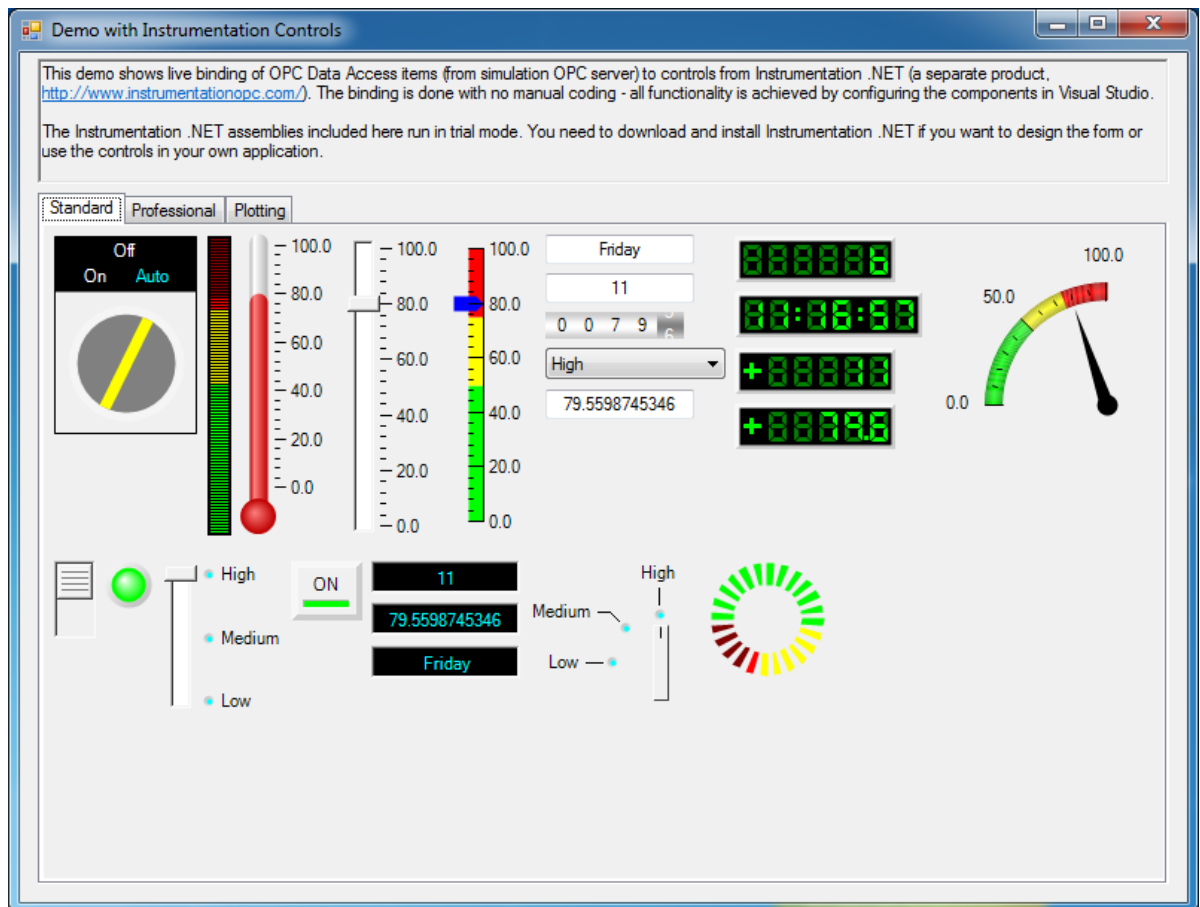
- Added **LiveBindingDemo** example: Shows live binding of OPC Data Access information (from simulation OPC server) to standard Windows Forms controls (Microsoft). All binding to OPC data is achieved with no manual coding, only by configuring the components in Visual Studio.



- Added **IndustrialGadgetsDemo** example: Shows live binding of OPC Data Access items (from simulation OPC server) to controls from Industrial Gadgets .NET (a separate product, <http://www.industrialgadgets.net/>). The application has been developed with no manual coding - all functionality is achieved by configuring the components in Visual Studio.



- Added **InstrumentationControlsDemo** example: Shows live binding of OPC Data Access items (from simulation OPC server) to controls from Instrumentation .NET (a separate product, <http://www.instrumentationopc.com/>). The binding is done with no manual coding - all functionality is achieved by configuring the components in Visual Studio.



- The screenshot displays the Symbol Factory .NET application window. The title bar reads "Demo with Symbol Factory .NET". The main content area contains a grid of various industrial symbols and controls, including:

 - Top row: A red circle, a blue motor, a black motor, a green tank, a factory with smoke, a laptop, a white box labeled "F1", a brown box, a stack of pipes, a long thin object, and a grey cylinder.
 - Second row: A red triangle, a monitor, a control panel, a yellow hand, a blue pin, a blue motor, a white bag, a blue tank, a green valve, and a green valve.
 - Third row: A grey tank, a grey motor, an orange cone, a grey box, a green pipe, a red valve, a blue gear, a grey box, a red truck, and a grey handle.
 - Fourth row: A yellow tank, a control panel, a grey box, a blue tank, a blue tank, a green valve, a green tank, a green tank, a green tank, and a map of the United States.
 - Fifth row: A grey motor, a grey motor, a grey motor, a grey motor, a grey motor, a grey motor, a grey motor, a grey motor, a grey motor, and a grey motor.

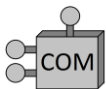
Below the grid, there is a text area with the following text:

This demo shows live binding of OPC Data Access items (from simulation OPC server) to controls from Symbol Factory .NET (a separate product, <http://www.symbolfactory.net/>). All controls are bound to a single source item. The application has been developed with no manual coding - all functionality is achieved by configuring the components in Visual Studio.

The Symbol Factory .NET assemblies included here run in trial mode, and only contain a limited set of symbols. You need to download and install Symbol Factory .NET if you want to design the form or use the symbols in your own application.

- Page 52 of 84

- Added **SimpleLogToSql** example: Logs OPC Data Access item changes into an SQL database, using a subscription.
- Added **LogAsStringToSql** example: Logs OPC Data Access item changes into an SQL database, using a subscription. Values of all data types are stored in a single NVARCHAR column.
- Added **LogAsUnionToSql** example: Logs OPC Data Access item changes into an SQL database, using a subscription. Values of all data types are stored in separate columns.
- Added **LogToSqlEnhanced** example: Logs OPC Data Access item changes into an SQL database, using a subscription. Item values and qualities are stored in their respective columns. Notifications with the same timestamp are merged into a single row.
- Added **WindowsService1** example: A Windows Service that subscribes to items from the simulation server, and logs their changes into a file.
- Added **OvenControl** example: Monitors sensors in an industrial oven, indicates level alarms by changing colors, allows the user to change a setpoint, and logs the values into a CSV file.
- Added **DataGridWebApplication** example: Demonstrates how easily can WebControls.GridView be populated with data read from OPC Data Access server.



- Added **VoleReadItemValue** example: Show how to read OPC item value using only C++ language features and libraries that are portable across compilers. Makes use of STLSoft and VOLE libraries.
- Added examples in T-SQL (for Microsoft SQL Server). **ReadAndDisplayValue.sql** reads and displays an OPC item value, using QuickOPC-COM.
- Added **ReadCurrentDataCom** example: Shows how to use QuickOPC-COM from managed code inside SQL Server (SQLCLR). Reads data of multiple OPC items and returns them in a recordset. The recordset can then be inserted into a log table in the SQL database.

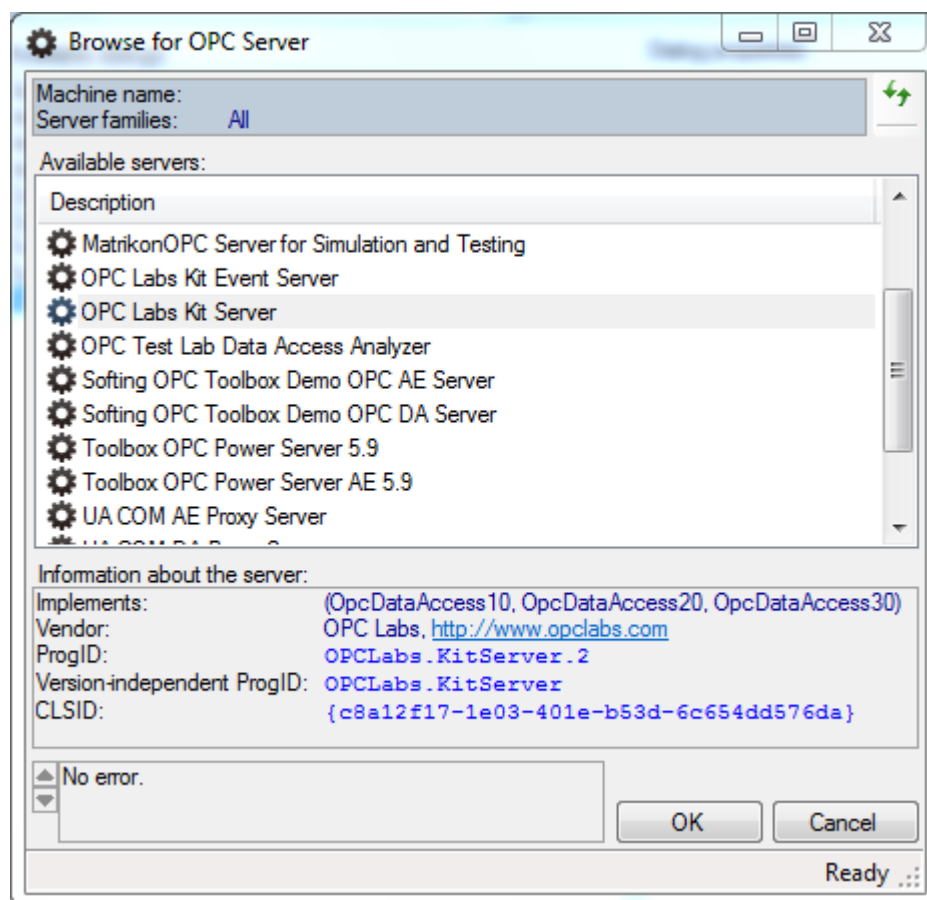
OPC Interoperability

- Improved browsing interoperability with RSLinx OPC Server.

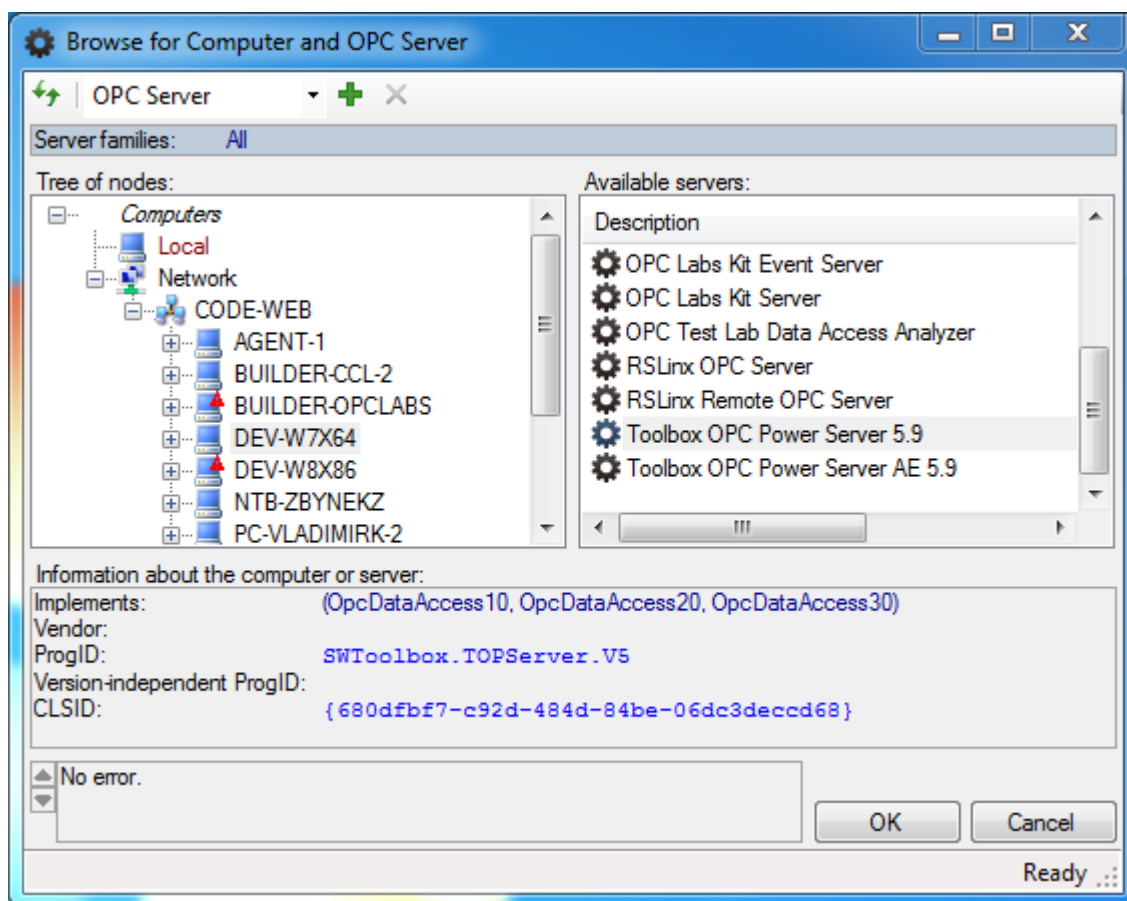
Details

Browsing Dialogs

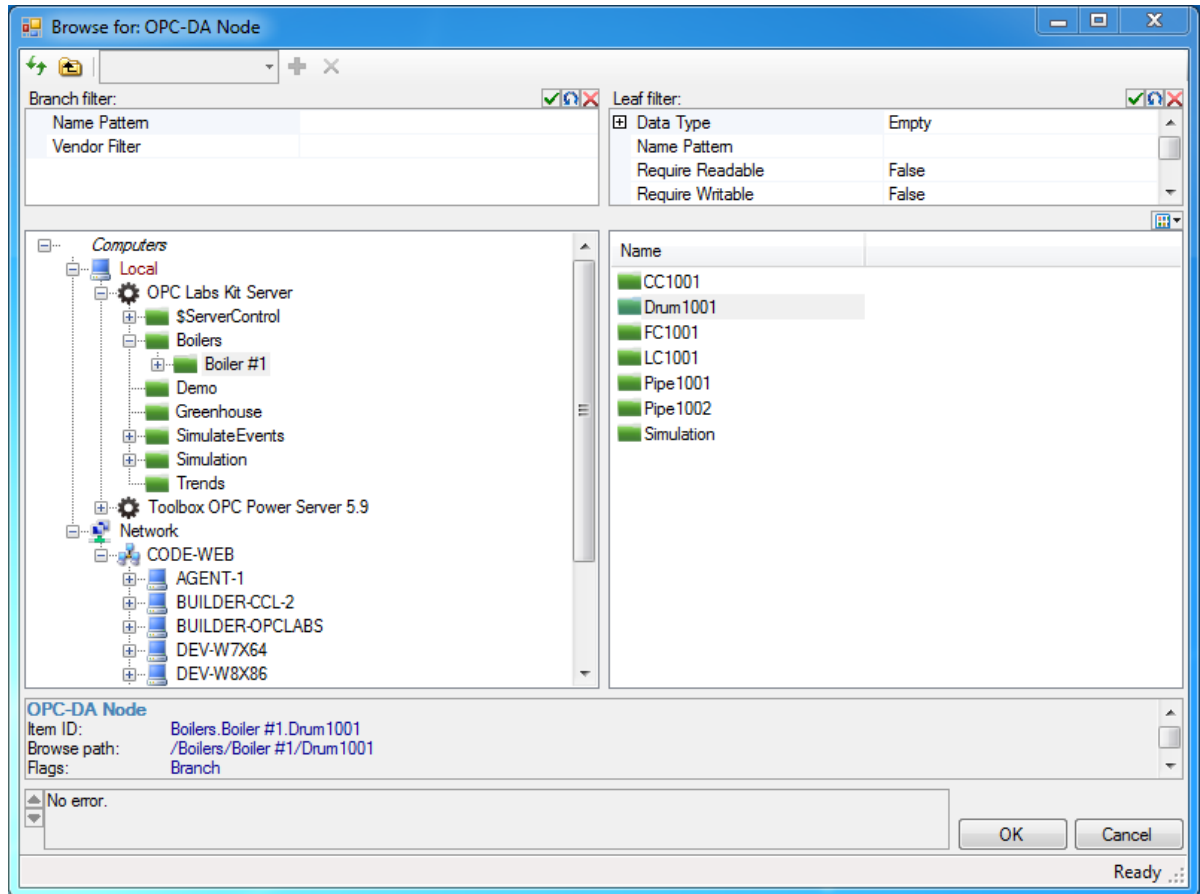
- **OpcServerDialog**: A dialog box from which the user can select an OPC "Classic" server.



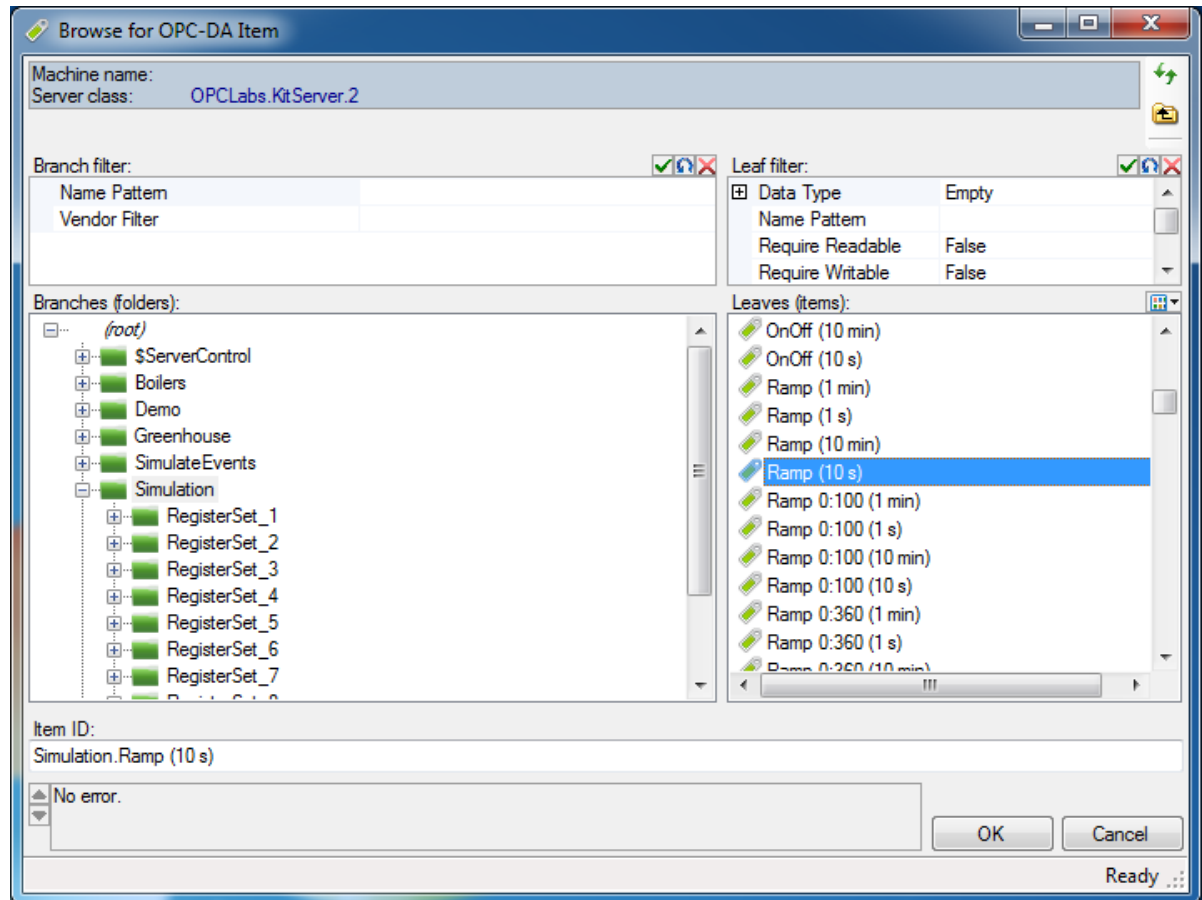
- **OpcComputerAndServerDialog**: A dialog box from which the user can select a computer and an OPC server residing on it.



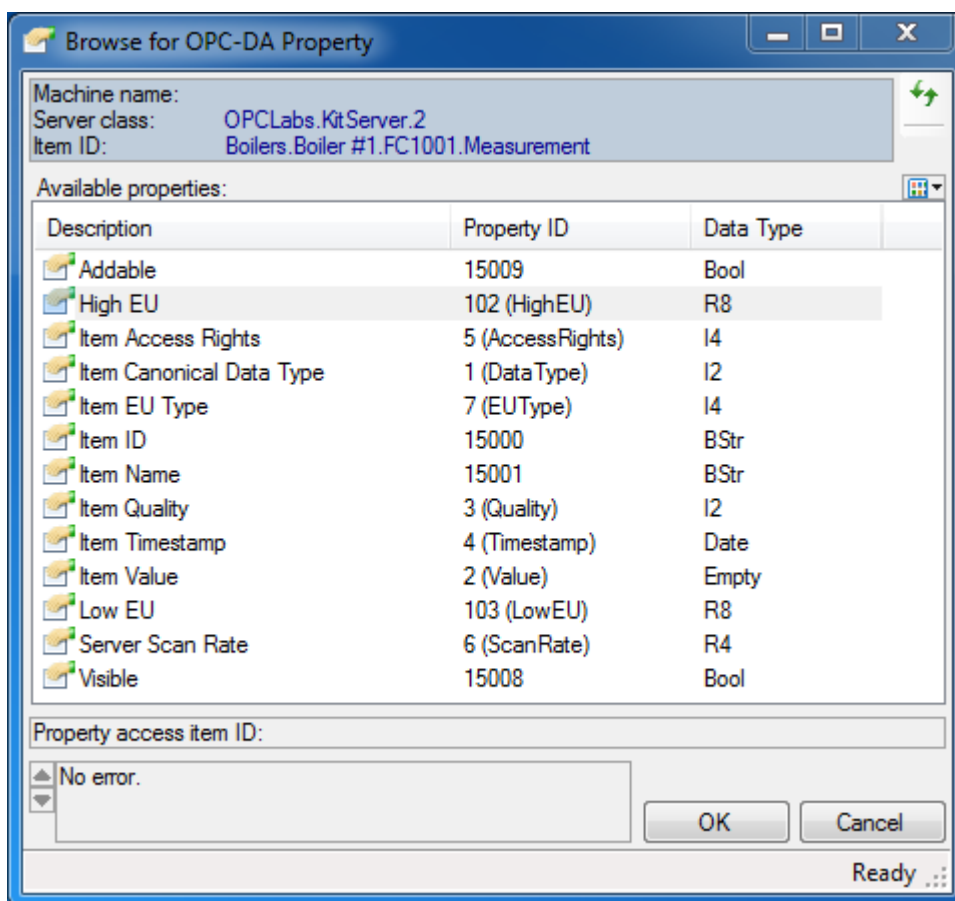
- **OpcBrowseDialog:** A dialog with various OPC nodes from which the user can select. This dialog can be configured to serve many different purposes.



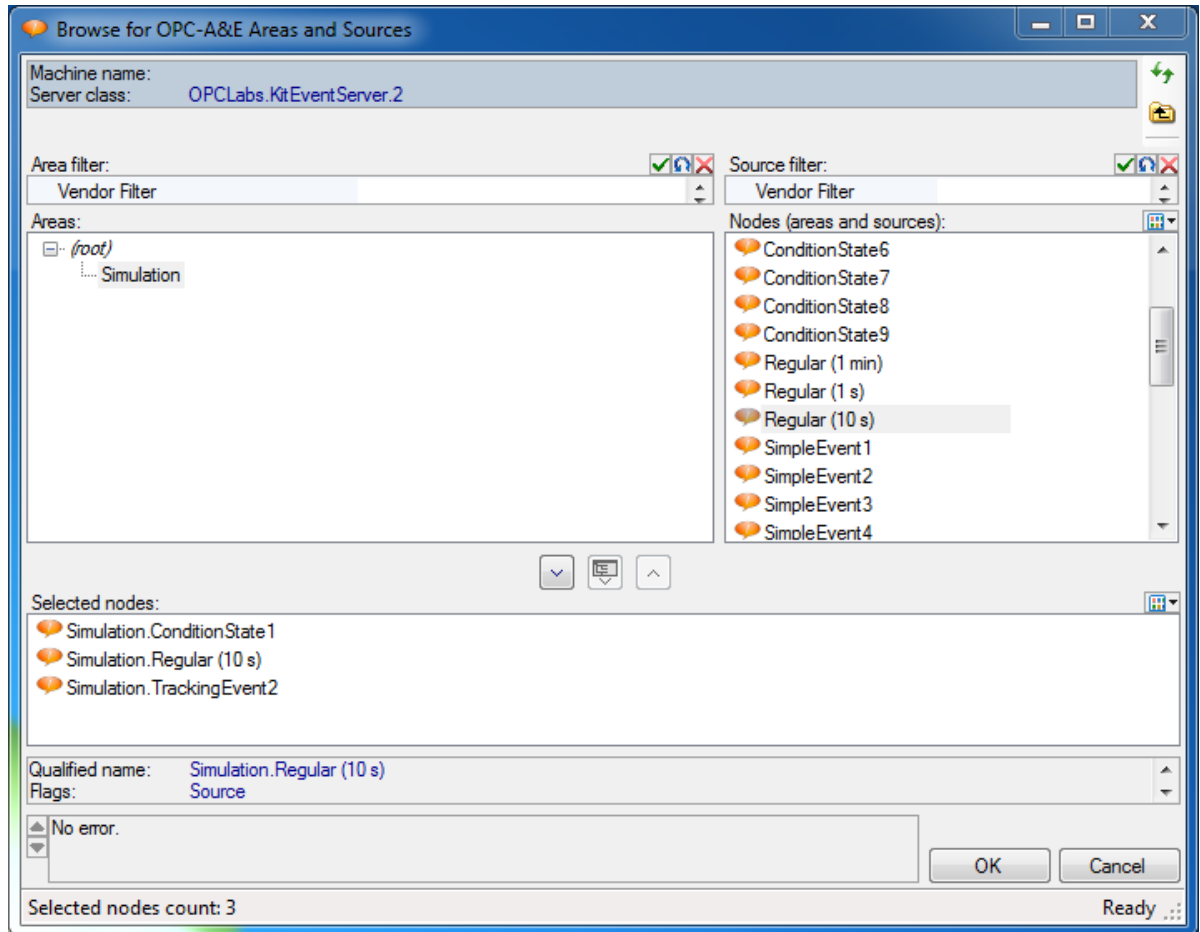
- **DAItemDialog**: a dialog box from which the user can select an OPC Data Access item.



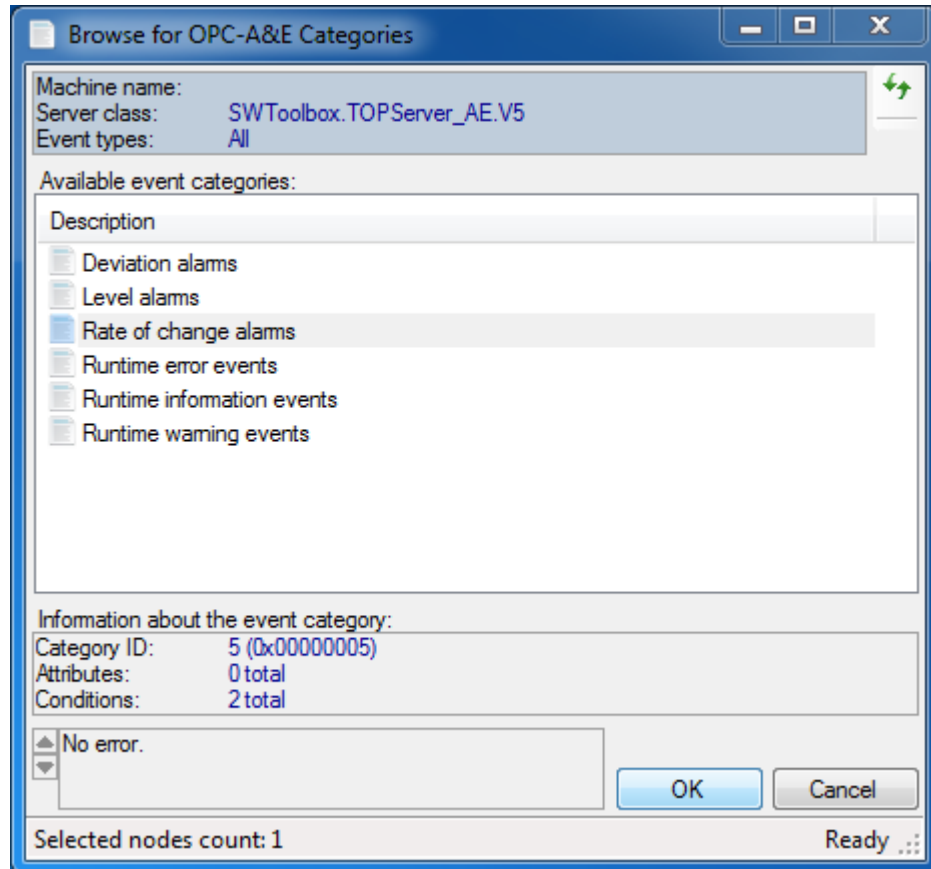
- **DAPropertyDialog:** A dialog box from which the user can select an OPC Data Access property.



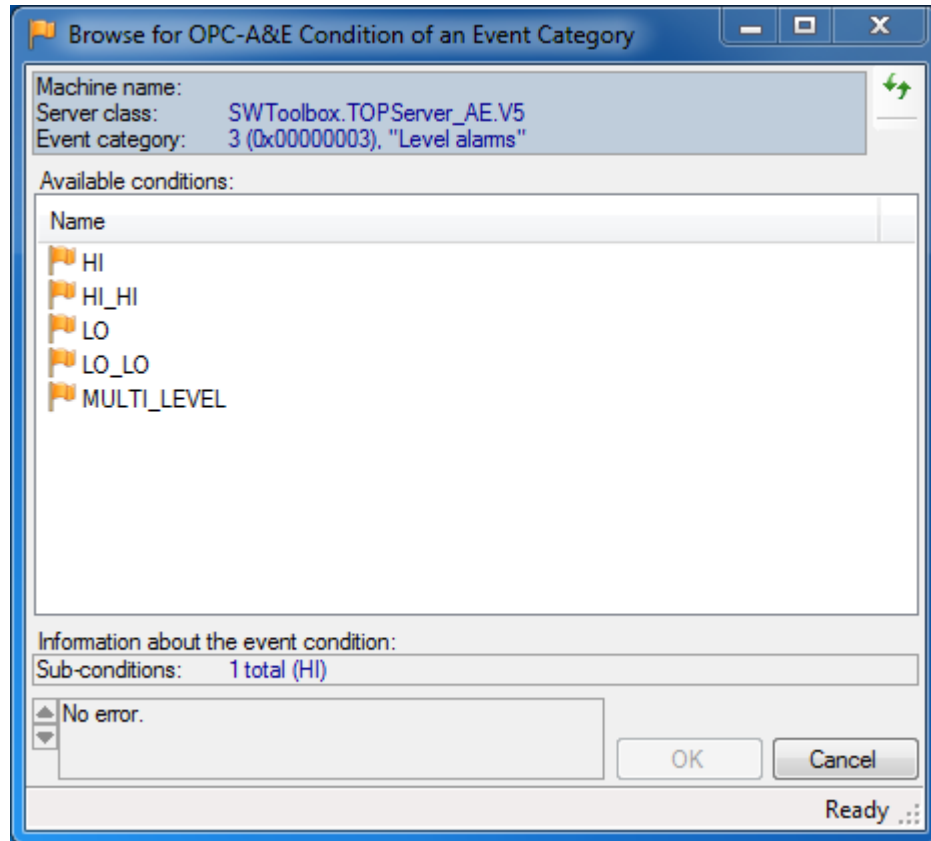
- **AEAreaOrSourceDialog:** A dialog box from which the user can select OPC-A&E event areas or sources.



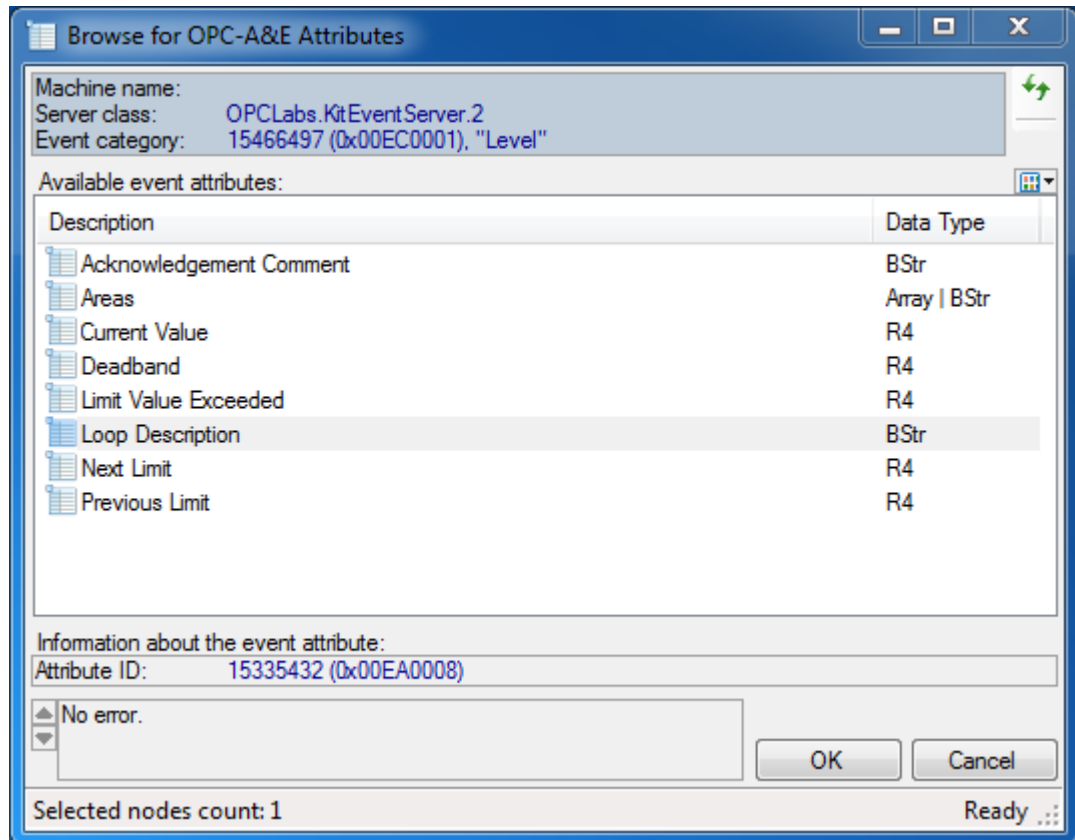
- **AECategoryDialog:** A dialog box from which the user can select from OPC-A&E event categories provided by the OPC server.



- **AECategoryConditionDialog**: A dialog box from which the user can select OPC-A&E category available on a specified event condition.



- **AEAttributeDialog**: a dialog box from which the user can select OPC-A&E event attributes.



What's New in QuickOPC-Classic 5.12

Key changes: Assorted Enhancements

Technology



- Minimum platform requirement is now unified to .NET Framework 3.5 Service Pack 1 (Client Profile or Full).
- In most scenarios, it is now possible to use the QuickOPC.NET assemblies from .NET Framework 4 CLR without extra precautions (i.e. without setting **useLegacyV2RuntimeActivationPolicy** in the configuration file).

Packaging



- Using a special technique, mixed-mode assemblies (containing native code, separate for x86 and x64 platforms) have been merged into a single assembly that appears as MSIL assembly to the consumers. You can now reference the same assemblies without regard for the target platform.
- Applications with QuickOPC.NET can now be built for “Any CPU” platform, without differentiating between 32-bit and 64-bit targets.
- The Visual Studio designer limitation (allowing only 32-bit components be loaded to the designer) no longer matters, also as result of the above assembly merging.
- Assemblies have been restructured (i.e. the product is now made of different assemblies as compared to Version 5.11), and assembly names have been made consistent, now always starting with “**Opclabs**” prefix.

Components



- Added possibility to use callback methods instead of (or in addition to) event handlers to **EasyDAClient** and **EasyAEClient** objects. This allows receiving notifications without hooking up event handlers, and directing them to different methods for different purposes easily. The callback methods can also be specified using anonymous delegates and lambda expressions, resulting in more concise and readable code.
- Added possibility to serialize and deserialize practically all objects (and their collections and dictionaries) using **Serializable** attribute and/or **ISerializable** interface. This serialization type is typically used with **BinaryFormatter** for storing objects in a binary format.

- Added possibility to serialize and deserialize practically all objects (and their collections and dictionaries) using **XmlSerializer** and **IXmlSerializable**. This serialization provides objects storage in XML format.
- Added **DefaultInstance** property to **EasyDAClient** and **EasyAEClient** objects. This property contains a default, shared instance of the client object, and allows writing shorter code, mainly for testing and non-library application code.
- Added **BrowsePath** property to **DANodeElement**. This property contains a string made of “short” item names starting from the root of the address space, up to the current node. Value of **ItemId** property of **DANodeElement** can now be a null reference; this can happen with some non-compliant OPC servers if they fail to return the full Item ID, typically for a branch.
- Added new **DANodeDescriptor** object which can be passed to **BrowseBranches**, **BrowseLeaves** and **BrowseNodes** and specifies the parent node for browsing. The **DANodeDescriptor** can describe the node either using the Item ID, or using the browse path, or both.
DANodeDescriptor objects can be easily created from strings (Item IDs will directly convert to **DANodeDescriptor**-s), or from **DANodeElement** objects that are returned from the browsing methods.
- Added **ErrorCode** property to **OperationResult** class, and to **EasyDAItemChangedEventArgs** and **EasyAENotificationEventArgs** classes. Application code can obtain the error code associated with the operation or event notifications without inspecting the **Exception** property.
- Added **Succeeded** property to **OperationResult** class.
- Added EasyOPC.NET Extensions (**EasyOpcClassicNetExtensions** assembly), with many new features (some listed below).
- All symbols have been annotated using ReSharper custom attributes for Code Analysis. This means that developers using ReSharper will immediately benefit from code inspection warnings, such as that possibly a null reference is given to an argument that must not be null, and others.
- Added named types for generic dictionaries returned by methods, and in some appropriate cases, changed them to be based on keyed collections instead. This is potentially a breaking change, mainly for applications that use browsing and querying, but the incurred code changes are not difficult. These changes have been made to comply with .NET recommendations and for serialization support.
- **SubscribeMultipleItems** method now returns simply an array of integers, not an array of **HandleResult**-s. The **HandleResult** class has been completely removed. This is a breaking change

for applications that use **SubscribeMultipleItems** method, but the incurred code change is not difficult, and the resulting code is smaller.

EasyOPC.NET Extensions

- For each primitive type, added type-safe methods that allow reading an item value already converted to the specified type, and type-safe methods that allow writing an item value with a specified type. Corresponding set of methods also exists for one-dimensional arrays of primitive types.
- For each primitive type, and one-dimensional arrays of primitive types, added type-safe methods that allow obtaining a value of an OPC property already converted to the specified type.
- Added methods that allow obtaining values of well-known OPC property directly, without specifying its property Id, and already converted to proper type.
- It is now possible to obtain all well-known OPC properties of an OPC item into a structure for easy access, in a single method call. It is also possible to obtain just a specified subset of OPC properties, and commonly used subsets come pre-defined with the component. It is also possible to combine the property sets (union operation).
- Added methods that obtain OPC properties of an OPC item into a dictionary, allowing easy retrieval of the property values without indexing an array.

OPC Interoperability



- Proper browsing for OPC nodes is now possible even with non-compliant OPC servers that fail to return Item IDs for branches.

Documentation and Help

- .NET: Help content now integrates with Microsoft Visual Studio 2008 Help (Microsoft Help 2 format).
- .NET: Help content now integrates with Visual Studio 2010 Help (Microsoft Help Viewer 1.0 format).
- .NET: Added a new chapter on EasyOPC.NET Extensions into the Concepts documents.
- Improved format of the Help.

Examples

- Added **SubscribeFromXml** example for .NET: Loads list of OPC items from an XML file and subscribes to them.
- Added **XmlEventLogger** example for .NET: Logs OPC Alarms and Events notifications into an XML file.
- Added **XmlLogger** example for .NET: Logs OPC Data Access item changes into an XML file.
- Added examples for EasyOPC.NET extensions.
- Added many other examples for .NET.
- COM: Added **ReadMultipleItems** and **WriteMultipleItemValues** examples in Visual C++.
- COM: Added example in Xbase++.

Internal Changes

- Significantly improved performance of **SubscribeMultipleItems**, **UnsubscribeMultipleItems**, **ChangeMultipleItemsSubscriptions** and **UnsubscribeAllItems** method with large number of items.
- The installation program now attempts to un-block the .CHM help files by registering them.

Bug Fixes

- .NET: Resolved a problem when placing a component onto a designer surface in a project targeting .NET Framework 4 generated an incorrect instantiation of **SynchronizationContext** that lead to inability to build initially, and a necessity of referencing the **WindowsBase** assembly.
- In QuickOPC.NET Reference, provided missing documentation to many elements.
- Properly included examples into QuickOPC.NET Reference.

What's New in QuickOPC-Classic 5.11

Key changes: Simplified Product Packaging

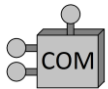
Packaging

- QuickOPC.NET and QuickOPC-COM setup have been merged into a single installation program, QuickOPC-Classic. The licensing has not been affected.
- All items from Bonus Pack installations have become optional parts of the core installation. When any of the “express” install options is selected, the Bonus Pack items are always included.
-
- All functionality from the **EasyOpcCommonNet** assembly has been moved into EasyOpcClassicNet assembly, and the **EasyOpcCommonNet** assembly has been removed. This means that referencing a single assembly, **EasyOpcClassicNet**, will suffice in many or most applications.
- Separate sets of assemblies for .NET Framework 4.0 (x86, x64) have been removed. Applications targeting .NET Framework 4.0 can reference the same assemblies as for earlier .NET Framework versions. Note that for this to work from .NET Framework 4, it is necessary to make a change to the application configuration file – please refer to “Quick Start” or “Concepts” document for details.
- Minimum platform requirements unified to .NET Framework 3.0.



Technology

- QuickOPC-COM is now officially supported on 64-bit operating systems (the components are 32-bit code still). Note that this is different from QuickOPC.NET which offers native 64-bit support.



OPC Interoperability

- Added error code/message definitions for all “classic” OPC specifications. Besides OPC Data Access and OPC Alarms and Events, error codes are now also translated for OPC Batch, OPC Command Execution, OPC Data Exchange, OPC Historical Data Access, and OPC Security (the extra definitions are useful when connecting to an aggregating server).

Documentation

- Documents for QuickOPC.NET and QuickOPC-COM have been merged into one, with icons differentiating parts that are specific to either technology.
- Documents in RTF (Rich Text Format) have been replaced by their equivalents in PDF format (Adobe Reader is required).

Examples

- Microsoft Visual Studio examples are now provided for Visual Studio 2008. They can all be automatically converted to Visual Studio 2010.
- With minor exceptions, all examples that existed in C# have been made in Visual Basic, and vice versa.

Installation

- The setup program now offers “Express install for .NET development”, “Express install for COM development”, or “Custom install” at the beginning of the wizard.
- The setup program checks for presence of Acrobat Reader (or any PDF Viewer), and warns if it is not present.
- Physical file layout has been rearranged.
- All 3rd-party redistributables that the developer may need are now installed into the “Redist” folder.
- Icons in Start menu have been rearranged.
- Added icon leading to Online Support.



What's New in QuickOPC.NET 5.10

Key changes: OPC Alarms and Events

OPC Interoperability

- Added support for OPC-A&E specification (Alarms and Events Custom Interface Standard Version 1.10). This specification is covered by an entirely new object hierarchy with its starting point at **EasyAEClient** class.

Components

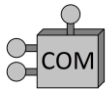
- Added new **SwtbExtenderReplacement** assembly. You can drop classes from this assembly into your application to replace the Software Toolbox Extender (www.opcextender.net) component.
- New **OpAlarmsAndEvents10** property in **ServerCategories** object.
- Added **EasyDAClient.MultipleItemsChanged** event. This event is capable of passing multiple item changes in one notification, improving the performance.
- Explicit item subscriptions now honor the requested update rate precisely (without attempting to merge the neighboring rates into "buckets"). This behavior can be controlled by new **EasyDATopicParameters.ExactManualGroupMatch** property.
- Minor members renaming took place to better capture the meaning (e.g. **EasyDAClientParameters.TopicRequestQueueSize** is now **RequestQueueSize**, and similarly **EasyDAClientParameters.TopicResponseQueueSize** became simply **ResponseQueueSize**).
- Some numeric error codes have changed (and have been added) to accommodate OPC Alarms and Events.

Packaging

- To facilitate the addition of OPC Alarms and Events support, some physical elements (such as assemblies and applications) have been renamed: **EasyOpcDANet** assembly becomes **EasyOpcClassicNet**, **EasyOpcDANetForms** assembly becomes **EasyOpcNetForms**, and **EasyOpcDANetDemo** application becomes **EasyOpcNetDemo**. Note that no logical elements (such as namespaces or classes) have been renamed for this reason.

Related Products

- Associated QuickOPC.NET Bonus Pack 5.10 has been created. A new project (CSharpDocExamples1) contains examples illustrating use of individual members from the reference documentation.



What's new in QuickOPC-COM 5.10

Key changes: OPC Alarms and Events

OPC Interoperability

- Added support for OPC-A&E specification (Alarms and Events Custom Interface Standard Version 1.10). This specification is covered by an entirely new object hierarchy with its starting point at **EasyAEClient** class.

Components

- New **OpcAlarmsAndEvents10** property in **ServerCategories** object.
- Added **EasyDAClient.MultipleItemsChanged** event. This event is capable of passing multiple item changes in one notification, improving the performance.
- Explicit item subscriptions now honor the requested update rate precisely (without attempting to merge the neighboring rates into “buckets”). This behavior can be controlled from the EasyOPC Options Utility, and the corresponding setting is called “exact manual group match”.
- Some numeric error codes have changed (and have been added) to accommodate OPC Alarms and Events.
- COM AppID, CLSIDs, IIDs, and version-dependent ProgIDs have changed.

Packaging

- To facilitate the addition of OPC Alarms and Events support, some physical elements (such as component executables, libraries and applications) have been renamed: **EOPCDAL.EXE** module becomes **EASYOPCL.EXE**, **EOPCDAI.DLL** module becomes **EASYOPCI.DLL**, and **EASYOPCDA.TLB** type library becomes simply **EASYOPC.TLB**. Note that no logical elements (such as namespaces or classes) have been renamed for this reason.

Related Products

- Associated QuickOPC-COM Bonus Pack 5.10 has been created. New examples, including examples for OPC Alarms and Events, have been added.





What's New in QuickOPC.NET 5.04

Key changes: 64-bit Development

Technology

- The components now allow development for x64 platform running in 64-bit mode (it was previously possible to run the programs developed with QuickOPC.NET on 64-bit systems, but only in 32-bit processes). Separate set of assemblies is provided for x64 development, however the interfaces remain identical. Development for x86 platform (32-bit) continues to be fully supported.
- For use on x64 platform, supporting software such as License Manager was also ported to 64-bits.

Installation

- The setup program now allows the user to select which platform support (x86, x64, or both) will be installed for development.
- On x64 platform, corresponding versions of Microsoft Visual C++ Redistributables, Demo application, License Manager, OPC Simulation Server, and other programs will be automatically chosen and installed.
- On x64 platform, corresponding version of OPC Core Components 3.00 Redistributables will be automatically chosen and installed (this includes proper OPC proxies and stubs).

Related Products

- New QuickOPC.NET Bonus Pack 5.04 has been created. Assemblies built for x64 platform have been added to the installation. All existing Visual Studio 2008 and Visual Studio 2010 examples have been enhanced by x64 platform configurations. The applications provided in binary form (OpcDAQQualityDecoder) are now available in x64 as well.



What's New in QuickOPC.NET 5.03

Key changes: Microsoft .NET Framework 4

Technology

- Added full support for Microsoft .NET Framework 4.0, and a separate set of assemblies compiled specifically for .NET Framework 4.0 is provided (.NET Framework 2.0, 3.0 and 3.5 remain supported, but not as primary development targets).
- Added full support for Microsoft Visual Studio 2010 (Visual Studio 2008 remains supported, but not as primary development target). Visual Studio 2010 can be used to target any version of .NET Framework with the product (2.0, 3.0, 3.5, or 4.0).

Installation

- The setup program now only checks and requires those prerequisites that are needed according to the set of components selected for installation.
- The setup program now fully automatically selects and installs dependency redistributable packages that are needed according to the set of components selected for installation.
- Installation of assemblies and a demo program targeting .NET Framework 2.0 is now turned off by default.

Related Products

- New QuickOPC.NET Bonus Pack 5.03 has been created. All existing examples have been converted to Visual Studio 2010 and retargeted to .NET Framework 4.0, with the use of new QuickOPC.NET assemblies (all pre-existing examples in Visual Studio 2008 have been retained, too).



What's New in QuickOPC.NET 5.02

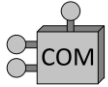
Key changes: OPC Universal Architecture Proxy

OPC Interoperability

- Added ability to communicate with OPC-UA (Universal Architecture) Servers through UA Proxy for OPC COM Clients. For instructions, see Advanced Topics -> OPC-UA (Universal Architecture) in the "Concepts" document.
- Added support for OPC Common 1.10 specification. This means that IOPCServerList2 interface can now be used to retrieve information about OPC servers when available, providing better compatibility and more detailed information about the OPC servers.

Technology

- Built with Visual Studio 2010. For highest compatibility, intentionally .NET Framework 2.0 is targeted, meaning that Visual Studio 2008 toolset is used when necessary. For all other projects (those that are not in .NET, e.g. License Manager or the Simulation OPC Server), Visual Studio 2010 toolset is used.



What's new in QuickOPC-COM 5.02

Key changes: OPC Universal Architecture Proxy

OPC Interoperability

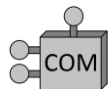
- Added ability to communicate with OPC-UA (Universal Architecture) Servers through UA Proxy for OPC COM Clients. For instructions, see Advanced Topics -> OPC-UA (Universal Architecture) in the "Concepts" document.
- Added support for OPC Common 1.10 specification. This means that IOPCServerList2 interface can now be used to retrieve information about OPC servers when available, providing better compatibility and more detailed information about the OPC servers.

Components

- The outgoing (source) interface of EasyDAClient object is now strictly a dispinterface, making it possible for languages such as PHP to consume events.

Installation

- Added (optional) installation of OPC UA COM Interop Components (including UA Proxy for OPC COM Clients, and UA Configuration Tool).
- In-process Server component is now turned off by default (Local Server remains turned on and becomes the default).



What's new in QuickOPC-COM 5.01

Key changes: Visual Studio 2010

Technology

- Built with Visual Studio 2010 toolset.



What's New in QuickOPC.NET 5.00

Key changes: Redesign for .NET

Technology

- All components are in managed code, i.e. purely .NET solution.
- Uses strongly typed parameters and properties.
- Designed in accordance with Microsoft recommendations for component development, and Code Analysis (FxCop) rules.
- There is no longer a COM process (middle layer) between your application and the OPC server.
- Windows Vista, Windows 7 and Windows Server 2008 support.
- Unicode is now used throughout the product.

Components

- Added ability to specify percent deadband with subscriptions.
- User interface components (dialogs) are now pure Windows Forms components.
- There is a new dialog for browsing for an OPC property.
- New method **ChangeItemSubscription** allows modification of subscription parameters without having to unsubscribe first.
- New **BrowseNodes** method combines capabilities of both **BrowseLeaves** and **BrowseBranches** methods.
- When browsing for OPC servers, version independent ProgId of the server can now be retrieved if available.
- The functionality of **OPCDAComponent** (in **OPCDAControls** assembly), providing event notifications on the UI thread for Windows Forms, has been replaced by the use of standard **SynchronizationContext**. This enables proper synchronization not only in Windows Forms, but in other environments as well, now and in the future.

- All methods are provided with set of useful overloads, simplifying their usage and allowing shorter and more readable code.
- Instead of specifying individual parameters for connection to OPC server, it is now possible to use a **ServerDescriptor** object. Similarly, parameters determining an OPC item can be passed using a **DAItemDescriptor** object. Use of these objects reduces the number of method arguments and makes the code shorter.
- Conversions from **ServerElement** and **DANodeElement** to **String** are provided, simplifying the use of browse results for further method calls.
- When browsing for OPC servers, a **ServerCategories** object is now provided, with information about specifications that the OPC server claims to support.
- New **VarType** value type makes it easy to specify data types used in OPC.
- New **DAQuality** value type manipulates OPC quality a whole, but also gives easy access to its individual bit fields. For display purposes, **DAQuality** can be directly converted to a string describing the quality.
- **DAVtq** object (holding value, timestamp and quality combination) can be converted to a string, fully describing contents of all its elements.
- New **DAAccessRights**, **DABrowseFilter**, **DADataSource**, **DANodeFilter**, and **DAPropertyId** types facilitate specifying the values using symbolic constants.
- A new **DAGroupParameters** object combines together parameter of a subscription.
- Method that deal with multiple elements at once return an array of **OperationResult** objects or objects derived from it, such as **ValueResult** or **DAVtqResult**. These objects contain error (exception) information as well as the actual result.
- Methods that deal with multiple elements at once can be passed a single argument, which is an array of objects derived from **OperationArguments**, simplifying argument preparation. An arbitrary **State** object can be used to hold application-defined information for each element, and this **State** object is then available in the **OperationResult** objects returned by the method call.
- All global parameters are accessible from your code as properties, and they are no longer stored in registry.
- OPC interoperability has been improved, based on tests with many OPC servers, and participation in OPC Interoperability Workshops.

- One-shot operations use a set of OPC groups separate from those that are created for subscription-based operations, resulting in more predictable behavior.
- Error messages have been improved and enhanced.

Documentation

- Detailed IntelliSense and Object Browser information is available.
- The reference documentation is now provided in standard Visual Studio (MSDN) layout.

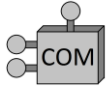
Packaging

- OPC Core Components that are redistributed with the product have been updated to a newer version.
- Examples, and other parts that do not directly belong to the core software, have been moved to a separate installation, QuickOPC.NET Bonus Pack.
- Components providing automation (COM) interfaces have been moved to a separate product installation, called QuickOPC.
- The Demo application, supplied with the product, has been enhanced to show many features of the product.
- Easier redistribution: Simply include the QuickOPC.NET assemblies with your application (Visual C++ Redistributables may also be needed).
- “Hidden” licenses are now being supported for OEM usage.

Removed

- Components providing Web service interfaces were removed, and will possibly be re-designed and become a separate product installation.
- Event Log Utility is no longer necessary and has been removed.
- Easy-DA Options Utility is no longer necessary and has been removed.
- OPC Item Generator utility has been removed.
- Support for operating systems older than Windows XP, or older than Windows Server 2003, has been removed.

- Methods that provide server-wide operations are now considered as “advanced”, and were removed.



What's new in QuickOPC-COM 5.00

Key changes: Redesign for Usability

Technology

- Windows Vista, Windows 7 and Windows Server 2008 support.
- Unicode is now used throughout the product.

Components

- Added ability to specify percent deadband with subscriptions.
- New method **ChangeItemSubscription** allows modification of subscription parameters without having to unsubscribe first.
- New **BrowseNodes** method combines capabilities of both **BrowseLeaves** and **BrowseBranches** methods.
- When browsing for OPC servers, version independent ProgId of the server can now be retrieved if available.
- When browsing for OPC servers, a **ServerCategories** object is now provided, with information about specifications that the OPC server claims to support.
- New **VarType** enumeration makes it easy to specify data types used in OPC.
- **DAVtq** object (holding value, timestamp and quality combination) can be converted to a string, fully describing contents of all its elements.
- New **DARedWriteMethod**, **DAAccessRights**, **DABrowseFilter**, **DADataSource**, and **DAPropertyId** enumerations facilitate specifying the values using symbolic constants.
- Method that deal with multiple elements at once return an array of **OperationResult** objects or objects derived from it, such as **ValueResult** or **DAVtqResult**. These objects contain error (exception) information as well as the actual result.
- Methods that deal with multiple elements at once can be passed an arbitrary **State** object can be used to hold application-defined information for each element, and this **State** object is then available in the **OperationResult** objects returned by the method call.

- Previously, methods that accepted **State** parameter allowed it in form of a string (BSTR) only. Any value (including objects) is now allowed in **State** parameters.
- The interfaces are now modeled after the .NET product, taking into account the Microsoft recommendations that also apply to COM.
- The generated events have uniform signature with “sender” and “event arguments” parameters, which reduces the coding needed to further process the event notifications in a generic way.
- OPC interoperability has been improved, based on tests with many OPC servers, and participation in OPC Interoperability Workshops.
- One-shot operations use a set of OPC groups separate from those that are created for subscription-based operations, resulting in more predictable behavior.
- **BrowseAccessPaths** method now returns array of strings, not a dictionary.
- Error messages have been improved and enhanced.

Tools and Instrumentation

- The EasyOPC-DA Options utility has been redesigned, reducing the number of pages to three.
- Added more useful settings to EasyOPC-DA Options utility, and allowed setting a special “Infinite” value wherever appropriate by simple check of a box.
- The default event logging settings now only log the most important events.

Documentation

- All interfaces, classes, properties and methods now have descriptive “help strings” in type libraries, making it possible for the development environments to provide friendly developer’s experience (such as IntelliSense).
- Brand new and comprehensive “Concepts” document.
- The reference documentation is now generated from the code and has been greatly enhanced.
- Code example for each method in a Reference.

Packaging

- OPC Core Components that are redistributed with the product have been updated to a newer version.
- Examples, and other parts that do not directly belong to the core software, have been moved to a separate installation, QuickOPC-COM Bonus Pack.
- .NET components have been moved to a separate product installation, called QuickOPC.NET.
- “Hidden” licenses are now being supported for OEM usage.

Removed

- Components providing Web service interfaces were removed, and will possibly be re-designed and become a separate product installation.
 - OPC Item Generator utility has been moved to QuickOPC Bonus Pack.
 - Support for operating systems older than Windows XP, or older than Windows Server 2003, has been removed.
 - Methods that provide server-wide operations are now considered as “advanced”, and were removed.
-