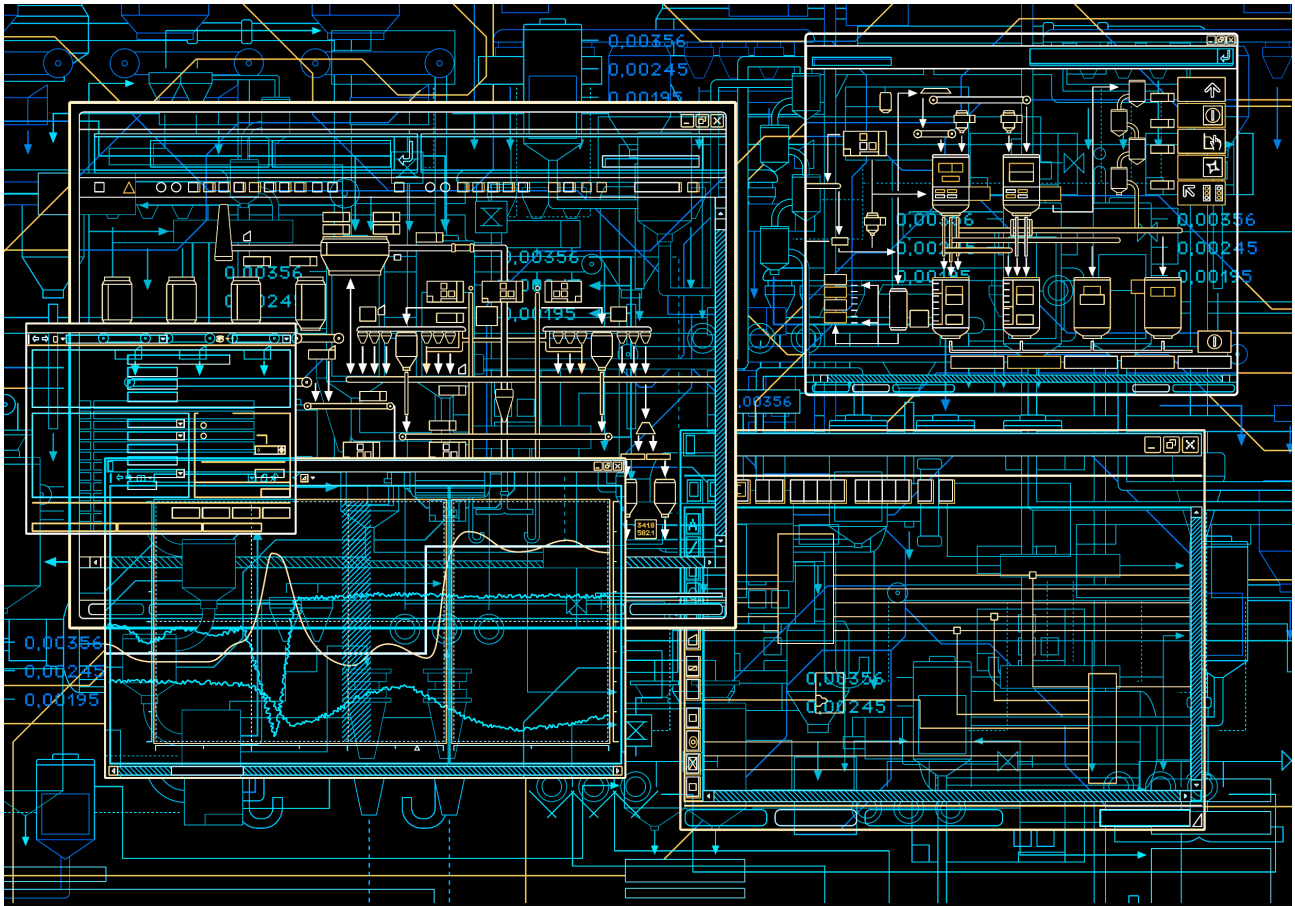

System 800xA

.NET Aspects Programmers Guide

System Version 5.1

Power and productivity
for a better world™

System 800xA

.NET Aspects Programmers Guide

System Version 5.1

NOTICE

This document contains information about one or more ABB products and may include a description of or a reference to one or more standards that may be generally relevant to the ABB products. The presence of any such description of a standard or reference to a standard is not a representation that all of the ABB products referenced in this document support all of the features of the described or referenced standard. In order to determine the specific features supported by a particular ABB product, the reader should consult the product specifications for the particular ABB product.

ABB may have one or more patents or pending patent applications protecting the intellectual property in the ABB products described in this document.

The information in this document is subject to change without notice and should not be construed as a commitment by ABB. ABB assumes no responsibility for any errors that may appear in this document.

In no event shall ABB be liable for direct, indirect, special, incidental or consequential damages of any nature or kind arising from the use of this document, nor shall ABB be liable for incidental or consequential damages arising from use of any software or hardware described in this document.

This document and parts thereof must not be reproduced or copied without written permission from ABB, and the contents thereof must not be imparted to a third party nor used for any unauthorized purpose.

The software or hardware described in this document is furnished under a license and may be used, copied, or disclosed only in accordance with the terms of such license. This product meets the requirements specified in EMC Directive 2004/108/EEC and in Low Voltage Directive 2006/95/EEC.

TRADEMARKS

All rights to copyrights, registered trademarks, and trademarks reside with their respective owners.

Release: December 2011
Document number: 2PAA107043-510 A

Table of Contents

Section 4 - Integration

Section 5 - Tutorial

About This User Manual

General

Any security measures described in this User Manual, for example, for user access, password security, network security, firewalls, virus protection, etc., represent possible steps that a user of an 800xA System may want to consider based on a risk assessment for a particular application and installation. This risk assessment, as well as the proper implementation, configuration, installation, operation, administration, and maintenance of all relevant security related equipment, software, and procedures, are the responsibility of the user of the 800xA System.

The Programmers Guide provides guidelines for development of aspect systems using .NET technology. The Programmers Guide includes the following sections:

Section 1, Introduction. Describes the development of aspect systems using .NET technology.

Section 2, Design Issues. Describes how to identify design issues while planning the development of aspect systems.

Section 3, Programming. Provides practical advice on .NET programming for development of an aspect system.

Section 4, Integration. Describes how to integrate the implementation of aspect systems, developed with .NET technology, into 800xA.

Section 5, Tutorial. Provides step-by-step instructions to create an aspect view using C#.

The support functionality for aspect development in .NET technology is available from 800xA 5.1 Feature Pack 1 onwards, and the required .NET Framework version is 3.5.

User Manual Conventions

Microsoft Windows conventions are normally used for the standard presentation of material when entering text, key sequences, prompts, messages, menu items, screen elements, etc.

Warning, Caution, Information, and Tip Icons

This User Manual includes Warning, Caution, and Information where appropriate to point out safety related or other important information. It also includes Tip to point out useful hints to the reader. The corresponding symbols should be interpreted as follows:

Electrical warning icon indicates the presence of a hazard that could result in ***electrical shock.***

Warning icon indicates the presence of a hazard that could result in ***personal injury***.

Caution icon indicates important information or warning related to the concept discussed in the text. It might indicate the presence of a hazard that could result in ***corruption of software or damage to equipment/property.***

Information icon alerts the reader to pertinent facts and conditions.

Tip icon indicates advice on, for example, how to design your project or how to use a certain function

Although Warning hazards are related to personal injury, and Caution hazards are associated with equipment or property damage, it should be understood that operation of damaged equipment could, under certain operational conditions, result in degraded process performance leading to personal injury or death. Therefore, fully comply with all Warning and Caution notices.

Terminology

A complete and comprehensive list of terms is included in ***System 800xA System Guide Functional Description (3BSE038018*)***. The listing includes terms and

definitions that apply to the 800xA System where the usage is different from commonly accepted industry standard definitions and definitions given in standard dictionaries such as Webster's Dictionary of Computer Terms. Terms that uniquely apply to this User Manual are listed in the following table.

Table 1. Associated Terms

Term	Description
Alarm Properties	An alarm is an abnormal state of a condition associated with an Aspect Object™. For example, the object FC101 may have the following conditions associated with it: Sensor error, flow over range, totalizer exceeded. An alarm is active as long as the abnormal state of the corresponding condition persists. An alarm is unacknowledged until a user acknowledges it. Alarm properties are global properties, which means that every object in the system have these properties. Global properties are defined on an aspect placed at the central place in the system, and not on every object. The ID of the aspect that provides global alarm properties is given as a constant in ***ABB.xA.Base.DataSubscriptionAspects.***
Application Programming Interface (API)	It serves as an interface between different software programs for interaction between them. This is similar to the way the user interface allows interaction between humans and computers. API can be created for applications, libraries, operating systems, and so on. It may include specifications for routines, data structures, and object classes.
Aspect	An aspect is the description of properties of an Aspect Object. Some examples of aspects are name, device management, DMS, and asset monitor.
Aspect Blob	For each aspect, data is stored in the aspect directory in a large binary object called blob. This is provided as a service directly from the system.

Table 1. Associated Terms (Continued)

Term	Description
Aspect Category	Specialization of an aspect type. For example, the Asset Monitors aspect type includes all of the Basic Asset Monitor aspect categories.
Aspect Object	A computer representation of real objects such as pumps and valves or a number of virtual objects such as service or object type. An Aspect Object is described by its aspects and organised in structures.
Aspect Object Type	Defines certain characteristics that are shared between several object instances, such as a basic set of common aspects. This makes it possible to create and efficiently re-use standardized solutions to frequently recurring problems. For example, rather than building an object from scratch for every valve in a plant, you can define a set of valve types, and then create all valve objects of these instances.
Aspect Persistent Data	Aspect state, stored persistently in the aspect directory.
Aspect Server	A server that runs the central functions of the Aspect Object architecture, such as Aspect Directory, Structure and Name Server, Cross Referencing, File Set Distribution, etc. Contains all Aspect Objects and their aspects.
Aspect System	A software system, which implements one or several aspect types by providing one or several aspect system objects.
Aspect Verbs	A framework function that offers custom functionality through the *IAfwAspectVerb* interface published through the *.add* file. The Aspect verb action appears in the aspect context menus.
Aspect View	Aspects can be presented in a number of ways depending on the task performed e.g. viewing or configuration. Each presentation form is called a view.

Table 1. Associated Terms (Continued)

Term	Description
Audit Messages	Audit messages are a logging of operator actions.
Component Object Model (COM)	A binary-interface standard introduced by Microsoft in 1993 that allows creation of software components. It is used to integrate custom applications and to enable inter-process communication and dynamic object creation in a large range of programming languages. The term 'COM' is often used in the Microsoft software development industry as an umbrella term that encompasses the OLE, OLE Automation, ActiveX, COM+ and DCOM technologies.
Context Menu	Appears when you right-click on an aspect object or an aspect. Lists aspect operations, actions, aspects, and global operations.
Node	A computer communicating on a network, for example, the Internet, Plant, Control, or I/O network. Typically, each node has a unique node address with a format, depending on the network to which it is connected.
Personal Computer (PC)	A computer running the Windows operating system.
Permission	A permission groups a set of operations that require the same authority. For each operation defined for an aspect or OPC property, the aspect category specifies the permission required to use that operation.

Table 1. Associated Terms (Continued)

Term	Description
Security	Security controls a user's authority to perform different operations on Aspect Objects, depending on several parameters: The user's credentials, as provided by Windows. The node where the user is logged in. This makes it possible to give a user different authority depending on where he/she is located, e.g. close to the process equipment, in a control room, or at home accessing the system through Internet. The object the user wants to perform the operation on.
Server	A node that runs one or several Services.
Structure	A hierarchical tree organization of Aspect Objects that describes the dependencies between objects. An Aspect Object can exist in multiple structures, for example, both in a Functional Structure and in a Location Structure.
System	System 800xA Base.
Windows Communication Foundation (WCF)	It is an application programming interface (API) in the .NET Framework for building connected, service-oriented applications.
Windows Presentation Foundation (WPF)	It is a software graphical subsystem in the .NET Framework for rendering user interfaces in Windows-based applications. WPF uses XAML to define and link various user interface elements.

Released User Manuals and Release Notes

A complete list of all User Manuals and Release Notes applicable to System 800xA is provided in ***System 800xA Released User Manuals and Release Notes (3BUA000263*)***.

System 800xA Released User Manuals and Release Notes (3BUA000263*) is updated each time a document is updated or a new document is released. It is in pdf format and is provided in the following ways:

- Included on the documentation media provided with the system and published to ABB SolutionsBank when released as part of a major or minor release, Service Pack, Feature Pack, or System Revision.
- Published to ABB SolutionsBank when a User Manual or Release Note is updated in between any of the release cycles listed in the first bullet.

A product bulletin is published each time ***System 800xA Released User Manuals and Release Notes (3BUA000263*)*** is updated and published to ABB SolutionsBank.

Section 1 Introduction

In the 800xA system, applications have to be built as aspect systems to be well-integrated into Process Portal A, so that these applications can participate in framework activities and use framework functions.

The integration platform in Process Portal A is based on COM technology, hence the aspects of Process Portal A must be implemented in terms of COM objects. To support the implementation of aspects in the .NET technology, a COM/.NET adapter has been introduced, which handles all the communication from the Process Portal A platform to the classes in the .NET assembly. See Figure 1. In this way, aspects can be implemented in .NET and the aspect developers need not be concerned with COM.

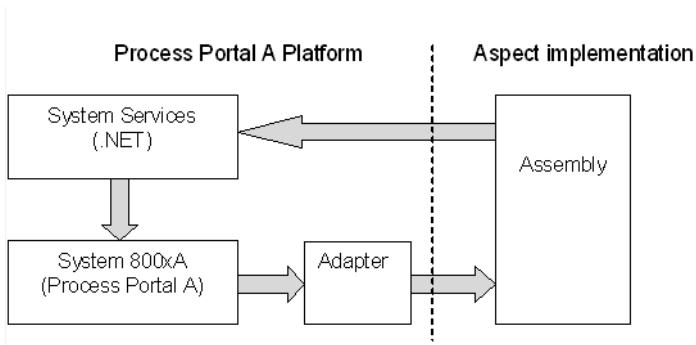

Figure 1. *.Net Aspect System Assembly Integration*

Aspects implemented with .NET technology have access to a limited set of operations for communication with System 800xA. This is available through a new, limited and safe .NET programming API (Application Programming Interface), which will be referred to as ***Services*** throughout this user manual.

The aspect can store data persistently in the aspect blob. This is provided as a service directly from the system.

The process of creating a new aspect type using .NET can be divided into two tasks:

1. Plan the functionality of an aspect and write an ***.add*** file. The ***.add*** file contains a description of the functionality offered by the aspect. For details, see Section 4, Integration.

2. Write the actual implementation code using Visual Studio and C#. For details, see Section 3, Programming.

Section 2 Design Issues

This section will help you to identify design issues while planning the development of an aspect system.

Conceptual Issues

Disposing of Objects

It is important to free resources that are not managed, such as files, streams, and handles, held by objects of the created classes; otherwise, the workplace may crash due to overload on the system memory. If there are objects that require disposal, call the ***dispose*** method on the aspect implementation classes in .NET. In an aspect view, objects should be disposed from the implementation of the ***Disconnecting*** method of the ***IAspectView*** interface. Inherit the classes from ***IDisposable*** and implement the ***dispose*** method. Ensure that the ***dispose*** call is propagated throughout the containment chain to free all the held resources.

For example, if an object A allocates an object B, and object B allocates an object C, then A's ***dispose*** implementation must call ***dispose*** on B, which must in turn call ***dispose*** on C. Objects must also call the ***dispose*** method of their base class if the base class implements ***IDisposable***.

Avoid using Microsoft standard interoperability in aspect implementation because it may cause memory and threading problems, leading to workplace crash.

Performance Issues

Persistent Data

Avoid reading/writing data frequently to the aspect persistent storage space, that is, more than once per second, depending on the data size. Aspect persistent data is designed for configuration data and for modestly sized data. Data sizes over 1 kilobyte are compressed automatically. Keep the data size as small as possible (that is, a few kilobytes). Examples in this manual, such as those under the topic Aspect Persistent Data on page 25, describe how to keep an updated copy of aspect persistent data in the aspect implementation classes.

Process Data Access

The subscription for process data is done through OPC DA. Avoid frequent subscribing and unsubscribing of the same properties because adding and removing OPC items are costly operations.

Limitations

The aspects developed in COM technology can implement a wide variety of framework functions, such as, aspect views, aspect verbs, and data subscription items. These aspects can also offer custom functionality through interfaces published through the ***.add*** file. At the time of writing, the aspects developed using .NET can implement only aspect views and aspect verbs.

Aspects developed in COM can send both system messages and audit messages. At the time of writing, aspects developed using .NET technology can send only audit messages.

Section 3 Programming

Introduction

The process of creating a new aspect type using .NET is divided into two main tasks:

- Writing the implementation code using Visual Studio and C#.
- Writing or updating an .add file with a new implementation binding.

Aspect implementation is based on attributes. There are attributes for associating a .NET class with an implementation binding and attributes for declaring methods as providers of functionality. State and behavior are added to an aspect type through implementation bindings. Implementation bindings are declared in .add files. To be associated with an implementation binding, a .NET class must have an implementation binding attribute. See example below. The GUID in the binding attribute should be the same as that of the binding definition in the .add file. For a description of .add files, see Section 4, Integration.

The following example shows the minimum implementation necessary for an aspect with a main view.

```
using System.Windows.Forms;
using ABB.xA.Base;
using ABB.xA.Base.Development;
[Binding("{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}")]
[AspectView(ViewId.Main, "Main view", UserRoleIndex.View)]
public partial class MainView: UserControl, IAspectView
{
    #region Construction
    public MainView()
    {
```

```
            this.InitializeComponent();
        }
        #endregion
        #region IAspectView Members
        public void Connecting()
        {
        }
        public void Disconnecting()
        {
        }
        public IAspectViewSite ViewSite { get; set; }
        #endregion
}
```

Signing

Implementation bindings have to be approved and signed by ABB before they can be deployed to a customer's system. Without a properly signed implementation binding, an aspect view receives a warning message, as shown in Figure 2.

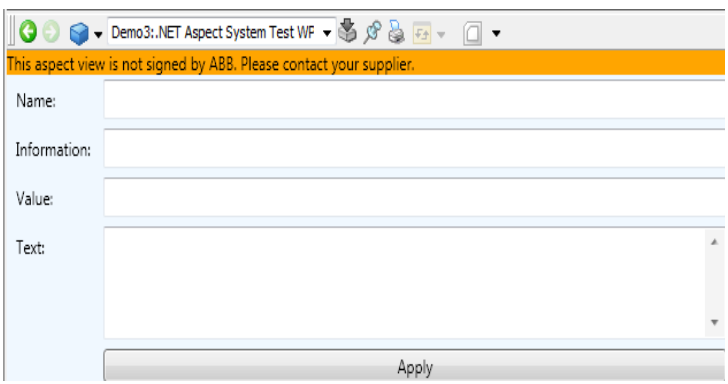

Figure 2. Aspect View Warning Message for Missing or Incorrect Signature

A signed implementation binding attribute has a second argument which is the signature.

```
[Binding("{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}",
"hjNAWh1jup7lGGVo+bFrj+GCdGQx7Ohi1PZOrFDqplJtafrgZ40KKaiL54mztfacXfOD0dyc
sFh9O+L9FjxQgw==")]
```

To become an approved supplier and to get implementation bindings signed, contact the product supplier responsible for Process Portal A.

System Access

Process Portal A is accessed programmatically through a set of services available in the ***ABB.xA.base*** assembly. Depending on the application, there are different methods to access the services.

A program (***.exe***) accesses the Base Services of the Process Portal A system through a high-level object of System 800xA. The program is then responsible for disposing the base services instance.

```
IDisposableBaseServices baseServices;
baseServices = System800xA.GetCurrent().GetBaseServices();
//...
baseServices.Dispose();
```

An aspect view accesses the services through its ViewSite. Since the ViewSite is owned by the workplace, the aspect view need not dispose the base services instance. Hence, the interface returned does not even have a ***dispose*** method.

```
IBaseServices baseServices = this.site.BaseServices;
```

Aspect Views

Aspect views are .NET controls implementing IAspectView from the ***ABB.xA***.Base assembly.

```
public partial class MainView: UserControl, IAspectView
```

Views can either be declared statically by adding the AspectView attribute and the Binding attribute to the .NET controls, or they can be declared dynamically in a view lister. The following example shows a statically declared aspect view:

```
[Binding("{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}")]
[AspectView(ViewId.Main, "Main view", UserRoleIndex.View)]
public partial class MainView: UserControl, IAspectView
```

A view lister is a static method annotated with the AspectViewLister attribute. The class containing the view lister must have a binding attribute, so that it is bound to the aspect through an implementation binding, as discussed in Section 4, Integration.

The parameter for the view lister method is an IAspectSite, which gives access to the system and to aspect persistent data. The return type is an enumeration of aspect view info. The following example shows how to declare an aspect view, named “X” dynamically.

```
[Binding("{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}")]
public class SomeClass
{
    [AspectViewLister]
    public static IEnumerable<AspectViewInfo> ListViews(IAspectSite site)
    {
        List<AspectViewInfo> views = new List<AspectViewInfo>();
        views.Add(new AspectViewInfo((ViewId)2, "X", UserRoleIndex.None,
        "TestAspectSystem.dll", "TestAspectSystem.ExtraView", true));
        return views;
    }
}
```

In this case the view implementation does not have to have any attributes. All that is needed is the IAspectView interface.

```
public partial class ExtraView: UserControl, IAspectView
```

Member variables of a class implementing a view lister cannot be used in the view lister method. To enforce this rule, the view lister method is made static. A user control can, for example, implement both an aspect view and a view lister method. The member variables of the user control cannot be used by the view lister implementation. A State can be shared through aspect persistent data only.

For more information on persistent data, see the topic Aspect Persistent Data on page 25.

Besides writing aspect view code, an operation must be added in the ***.add*** file. In the following example, the added operation, shown in bold face, is ***IAfwAspectViewControl*** that tells the system that the implementation binding also publishes aspect views. The other operation shown in the example is ***IAfwAspectVerb*** for aspect verbs.

```
implBind
{
    id = "{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}"
    aspectId = "{7E4CD9F9-93FF-11D2-86B3-0000F87884BE}"
    name = Example
    clsid = "{1AAA2CB0-1EF1-4996-A987-B37604ADFEFC}"
    assembly = "TestAspectSystem.dll"
    operations {
          "{635D2C6F-1EA6-4D7B-9736-C011CB170C54}"
          "{C4B81CDA-3C4A-11D2-A1E0-0000F8788595}"
    }
}
```

An aspect view accesses the system services through its ***ViewSite*** reference. To receive the reference, the aspect view must implement the interface ***IAspectView*** from the ***ABB.xA.Base*** assembly.

This interface has the following members:

- `void Connecting ()`

  Called after the aspect view gets its ***ViewSite*** reference. Only after this point, you can start accessing the system, connect event handlers, and read the persistent data. Do not use alternatives, like the user control's Load event.

- `void Disconnecting ()`

  Called before the aspect view is closed. The user control is disconnected but not disposed at this point. Use this method to release the resources immediately after the aspect view is closed.

- `IAspectViewSite ViewSite { get; set; }`

  Aspect ViewSite is the aspect view's connection to the system. Through ViewSite, it can access the system functions, and get notifications. The ***ViewSite*** property is set by the workplace before ***Connecting*** is called.

The Aspect ViewSite has the following members:

- `AspectRef Aspect { get; }`

  Specifies the identity of the aspect for which this object implements a view.

- `IBaseServices BaseServices { get; }`

  Denotes a reference to the base services that give access to Aspect and object-related operations.

- `IWorkplaceServices WorkplaceServices { get; }`

  Denotes a reference to the workplace services that offer workplace functions, like navigation and context menu.

- `object GetNavigationParameter(string key);`

  If the source of a navigation sends parameters, then the parameters can be read in the destination view, using the ***GetNavigationParameter*** method of the ViewSite. Parameters are key/value pairs where the key is a parameter name.

  ```
  event EventHandler AspectDeleted;
  event EventHandler<AspectChangedEventArgs> AspectChanged;
  ```

These two events are notifications regarding the aspect.

If the persistent data (blob) or any other state has changed, then the ***AspectChanged*** event is raised. The event argument states the reason for the notification.

If the aspect is being deleted, then the ***AspectDeleted*** event is raised. At this point, the aspect no longer exists inside the system, and the persistent aspect data cannot be read or written.

Aspect Persistent Data

Designing aspects using COM technology involves developing a View ASO and a Data ASO.

A Data ASO is a COM object that keeps unpacked aspect data and exposes interface methods to manipulate the data. The aspect view communicates with the Data ASO to get data for the view, and to write changes.

Developing an aspect in .NET does not involve a Data ASO. Data access is available as a service in the API.

The aspect view should maintain an updated unpacked copy of the aspect data. When a user modifies the aspect view, the modifications should be stored locally until the user decides to apply the changes. Only after applying the changes, the data should be written to the Aspect Directory.

The view site has a ***GetAspectData*** method to get and a ***SetAspectData*** method to set aspect persistent data in the aspect blob. This data is stored in the aspect blob in a compressed text format.

```
this.myData = this.ViewSite.GetAspectData<MyData>();
this.ViewSite.SetAspectData<MyData>(this.myData);
```

GetAspectData and ***SetAspectData*** operate on a user-defined data class. The data class is defined using WCF data contracts. The class and the persistable members are annotated with attributes for serialization.

```
using System.Runtime.Serialization;

[DataContract(Name = "MyData")]
public class MyData
{
    [DataMember]
    public string Name { get; set; }
}
```

A minimum aspect implementation keeps at least a member of the data class and a Boolean to indicate whether the data is changed locally or not.

```
public partial class MyConfigView: UserControl, IAspectView
{
    #region Data
    private bool modified = false;
    private MyData myData;
    #endregion
}
```

When the aspect view is first shown, the persistent data is read to the local data object. An event handler is set up on the ‘Aspect Changed’ event so that the data can be refreshed, as required.

```
public void Connecting()
{
    // Connect an aspect changed event handler
    this.ViewSite.AspectChanged += new
    EventHandler<AspectChangedEventArgs>(ViewSite_AspectChanged);

    // Read persistent data
    this.myData = this.ViewSite.GetAspectData<MyData>();
    // Update UI
    this.txtName.Text = this.myData.Name;
}
```

When an aspect change is detected, the data can be refreshed.

```
void ViewSite_AspectChanged(object sender, AspectChangedEventArgs e)
{
    // We are only interested in persistent data changes
    if ((e.Details & AspectChangeDetails.Blob) != 0)
        {
           // Aspect has changed, read persistent data.
           this.myData = this.ViewSite.GetAspectData<MyData>();
           // Update UI
           this.txtName.Text = this.myData.Name;
           this.modified = false;
        }
}
```

Aspect data is saved when the **Apply** button is clicked.

```
private void btnApply_Click(object sender, EventArgs e)
{
    // Update data object from UI
    this.myData.Name = this.txtName.Text;
    // Save persistent data
    this.ViewSite.SetAspectData<MyData>(this.myData);
    this.Modified = false;
}
```

Before the aspect view is closed, any unsaved local changes in aspect data have to be saved persistently.

```
public void Disconnecting()
{
    if (this.modified)
    {
        // -- Optionally, launch a "Save changes YES/NO" dialog here

        // Update data object from UI
        this.myData.Name = this.txtName.Text;
        // Save persistent data
```

```
            this.ViewSite.SetAspectData<MyData>(this.myData);
        }
}
```

Data Versioning

The persistent data classes support versioning through the WCF (Windows Communication Foundation) standard. For example, if a new data member is required, a new class with the same data contract name is defined and used instead of the original class. In the following example, a new member ***Information*** is added.

```
[DataContract(Name = "MyData")]
public class MyData2
{
    [DataMember]
    public string Name { get; set; }

    [DataMember]
    public string Information { get; set; }

    [OnDeserializing]
    void OnDeserializing(StreamingContext context)
    {
        // Set default values before deserializing
        this.Information = "Default information";
    }
}
#region Data
private bool modified = false;
private MyData2 myData;
#endregion
```

If an aspect blob containing serialized data from ***MyData*** is read into ***MyData2*** object, then the Information member of ***MyData2*** gets the default value "Default information".

```
this.myData = this.ViewSite.GetAspectData<MyData2>();
```

Navigation

Navigation is a command from an operator to the workplace to show an aspect view. The workplace configuration decides where the aspect view should be displayed.

Navigation from an aspect view to another aspect view is available through a set of ***navigation*** methods from Workplace Services. The aspect view gets Workplace Services from its View Host. The following example shows navigation to the default aspect of an object defined by its ID.

```
Dictionary<string, object> parameters = new Dictionary<string, object>();
parameters.Add("Message", "Kalle Kula");
this.ViewSite.WorkplaceServices.Navigate(objId, parameters);
```

The ***navigation*** parameter message can be read by the target view. Only strings and value types can be sent as a parameter.

The target aspect view can read navigation parameters through a method on the ViewSite.

```
string parameter = (string)site.GetNavigationParameter("Message")

if (parameter != null)
    label1.Text = (string)parameter;
else
    label1.Text = "No parameter supplied!
```

Context Menu

The system context menu for an aspect or an object from any aspect view is a key feature of System 800xA. An aspect view implemented in .NET technology can display the system context menu through Workplace Services.

```
this.ViewSite.WorkplaceServices.ShowContextMenu(objId);
```

Workplace information

From an aspect view, it is possible to listen for events from panes in the workplace. If the contents of a pane is changed or if a content is closed, a ***PaneNotification*** event is raised. The following example shows how to monitor when the contents of the preview area of the workplace is replaced. The example also shows how to retrieve information about a specific pane.

```
{
    IWorkplaceServices ws = this.ViewSite.WorkplaceServices;
    // Set up an event handler
    ws.PaneNotification += new
    EventHandler<PaneNotificationArgs>(WS_PaneNotification);
    // Subscribe for events from the pane with name idPreView
    ws.SubscribeForPaneNotifications("idPreView");
}
// Event handler that shows the URL of the new contents
void WS_PaneNotification(object sender, PaneNotificationArgs e)
{
    if (e.Reason == PaneNotificationReason.ChangedContent)
    {
        PaneInfo pi = ws.GetPaneInfo(e.PaneName);
        MessageBox.Show(pi.OriginalURL);
    }
}
```

Process Data Access

The Process data from the system can be read through ***IDataSubscriptionServices***. The values are read by setting up a subscription for value changes on data properties. The properties are represented by a Property Reference class consisting of an aspect reference and a property name. After subscribing for properties, an event with the current values arrives immediately after the call for all the properties in the subscription. After this, only property changes will generate an event. When change notifications are no longer required for a collection of properties, the subscription can be stopped by unsubscribing the properties through ***IDataSubscriptionServices***.

The following example shows how to subscribe for ***IsAlarmActive*** property value from an object. Alarm properties are global properties, which means that every object in the system have these properties. Global properties are defined on an aspect placed at the central place in the system, and not on every object. The ID of the aspect that provides global alarm properties is given as a constant in ***ABB.xA.Base.DataSubscriptionAspects.***

```
using GlobalAlarmAspects = ABB.xA.Base.Constants.AlarmAndEvent.Aspects;
IDataSubscriptionServices ds =
    this.ViewSite.BaseServices.DataSubscriptionServices;
// Hook up an event handler for data changes
ds.DataChanged += new
    EventHandler<DataChangedEventArgs>(DSServices_DataChanged);

// Subscribe for the IsAlarmActive property on my object
AspectRef alarmProperties =
    new AspectRef(this.ViewSite.Aspect.ObjectId,
        GlobalAlarmAspects.GlobalAlarmProperties);

SubscriptionResult res = ds.Subscribe(
    new PropertyReference(alarmProperties, "ISALARMACTIVE"));

if (res.MasterResult == SubscriptionMasterResult.ItemError)
    … take care of item errors

// Event handler for value changes for properties that
// have been subscribed for
void DSServices_DataChanged(object sender, DataChangedEventArgs e)
{
    foreach (var value in e.Values)
    {
        … process values
    }
}
```

System Messages

Audit messages can be sent from aspects developed in .NET. To send an audit message, a suitable system message is required. System messages and message sources can be defined for the product using ***AfwSysMsgDefineTool*** AfwSysMsgDefineTool.exe located in the ***Process Portal A\bin*** directory.

The output is a ***.mdd*** file that must be placed in the ***\mdd*** directory of your product, for example, ***C:\<your_product_name>\mdd***

Figure 3 shows a message definition of the message class Operator Action audit event. This message has two parameters, ***Name*** and ***Text***. These two parameters must be supplied by the sender of the message.

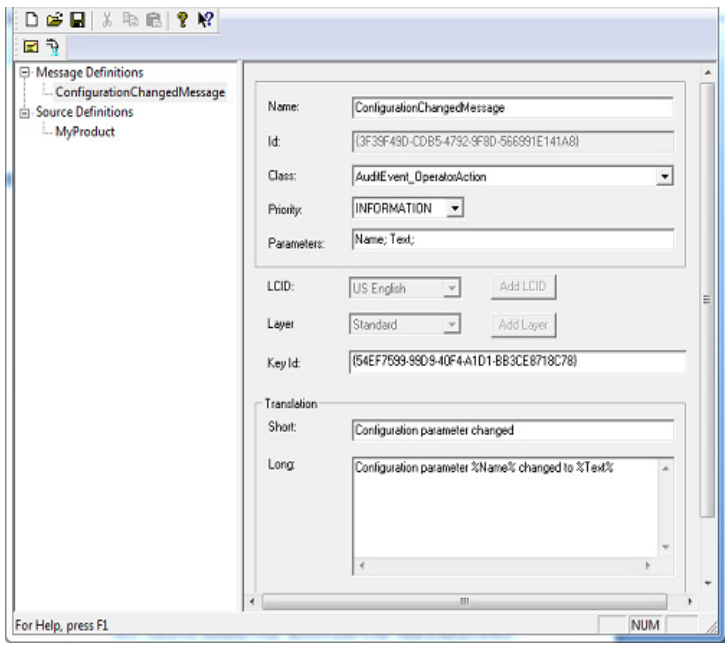

Figure 3. Message Definition Tool

Audit messages can be sent through Security Services. The following example shows how an audit event is sent, based on a system message, with two embedded parameters, ***Name and Text.***

```
this.ViewSite.BaseServices.SecurityServices.GenerateAuditEvent(
    this.ViewSite.Aspect,
    AuditTransactionType.ModifyConfiguration,
    true, // Postpone until transaction completed
    new MessageSourceId("{4FDD4A63-95D1-4963-8983-A5ABD7D03579}"),
    new MessageId("{54EF7599-99D9-40F4-A1D1-BB3CE8718C78}"),
    new SystemMessageParameter("Name", this.myData.Name),
    new SystemMessageParameter("Text", this.myData.Text));
```

Aspect Verbs

Aspect verbs and object verbs are actions that appear in the Aspect and Object context menus. Aspect verbs and/or object verbs are either implemented as a static methods on any class in the assembly, or defined dynamically in a verb controller. The static method that implements a verb should be annotated with the ***AspectVerb*** attribute from the ***ABB.xA.Base*** assembly. The class must also have a binding attribute, so that it is bound to the aspect through the implementation binding, as discussed in Section 4, Integration.

```
/// <summary>
/// Verb id's are integer values, and it is recommended to use an fn
/// enumeration for the values.
/// </summary>
enum VerbID
{
    TheVerb,
    AnotherVerb
}

/// <summary>
/// This is an example implementation of an Aspect & object verb.
/// </summary>
[Binding("{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}")]
public static class MyVerbs
{
    [AspectVerb(
```

```
            VerbID.TheVerb,
            "The Verb",
            ContextMenuContext.Object | ContextMenuContext.Aspect,
            UserRoleIndex.Operate)]
        public static void TheVerb(IAspectVerbSite site)
        {
            IAspectObjectsServices aos;
            aos = site.BaseServices.AspectObjectsServices;
            ObjectInfo info = aos.GetObjectInfo(site.Aspect.ObjectId);
            MessageBox.Show("The verb says: Hi I'm" + info.Name);
        }
}
```

Besides writing the verb method code, an operation must be added in the .add file. In the following example, the added operation, shown in bold face, is ***IAfwAspectVerb***, that tells the system that the implementation binding also publishes verbs. The other operation shown in the example is ***IAfwAspectViewControl*** for aspect views.

```
implBind
{
    id = "{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}"
    aspectId = "{7E4CD9F9-93FF-11D2-86B3-0000F87884BE}"
    name = Example
    clsid = "{1AAA2CB0-1EF1-4996-A987-B37604ADFEFC}"
    assembly = "TestAspectSystem.dll"
    operations {
           "{635D2C6F-1EA6-4D7B-9736-C011CB170C54}"
           "{C4B81CDA-3C4A-11D2-A1E0-0000F8788595}"
    }
}
```

Member variables of a class implementing a verb cannot be used in the verb method. To enforce this rule, the verb methods are made static. A user control can, for example, implement both an aspect view and a verb method. The member variables of the user control cannot be used by the verb implementation. A State can be shared through aspect persistent data only. For more information on persistent data, see the topic Aspect Persistent Data on page 25.

The default appearance of a declared aspect verb is ***visible***, ***enabled*** and ***unchecked***. To change the default appearance before the context menu is shown, implement a static method annotated with the ***AspectVerbControl*** attribute. The parameter for this method is an ***IAspectVerbControlSite,*** which in addition to giving access to the system and to aspect persistent data, also gives access to the portion of the context menu which is reserved for verbs. This method will be called just before the context menu is shown.

The verb control site has the following members

- `AspectRef Aspect { get; }`

  Specifies the identity of the aspect .

- `IBaseServices BaseServices { get; }`

  Denotes a reference to the base services that give access to Aspect and object-related operations.

- `SubMenu Verbs { get; }`

  Denotes a reference to the context menu portion reserved for aspect verbs.

- `T GetAspectData <T>();`

  Get aspect persistent data

- `void SetApectData<T> (Tdata);`

  Set aspect persistent data

The following example illustrates how to control the enable state of a verb by implementing a verb controller.

```
/// <summary>
/// This is an example implementation of an Aspect & object verb.
/// The AspectVerbControl is optional and should only be implemented
/// if you want to control visibility and enable state of each
/// individual verb, or if you want to specify new verbs dynamically.
/// </summary>
[Binding("{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}")]
public static class MyVerbs
{
    [AspectVerbControl]
    public static void VerbControl(IAspectVerbControlSite site)
{
        // Make the verb "TheVerb" appear grayed.
        // The verb id from the verb gets an offset internally of 1000.
        site.Verbs.Item(1000).EnableState = MenuItemEnableState.Grayed;
}
    [AspectVerb(
        VerbID.TheVerb,
        "The Verb",
        ContextMenuContext.Object | ContextMenuContext.Aspect,
        UserRoleIndex.Operate)]
    public static void TheVerb(IAspectVerbSite site)
    {
        IAspectObjectsServices aos;
        aos = site.BaseServices.AspectObjectsServices;
        ObjectInfo info = aos.GetObjectInfo(site.Aspect.ObjectId);
        MessageBox.Show("The verb says: Hi I'm" + info.Name);
    }
}
```

The following example shows how to use the verb controller to add a verb with two submenu items.

```
using XADEV = ABB.xA.Base.Development;


[AspectVerbControl()]
public static void VerbControl(IAspectVerbControlSite site)
{
    XADEV.MenuItem topLevelItem = new XADEV.MenuItem(
        2000, "My Verb", ContextMenuContext.Object);
    SubMenu cascadeMenu = new SubMenu();
    cascadeMenu.Add(new XADEV.MenuItem(
        2001, "SubItem1", ContextMenuContext.Object));
    cascadeMenu.AddSeparator(ContextMenuContext.Object);
    cascadeMenu.Add(new XADEV.MenuItem(
        2002, "SubItem2", ContextMenuContext.Object, null,
        MenuItemEnableState.Grayed, MenuItemCheckedState.Checked,
        UserRoleIndex.None));
    topLevelItem.SubMenu = cascadeMenu;
    site.Verbs.Add(topLevelItem);
}
```

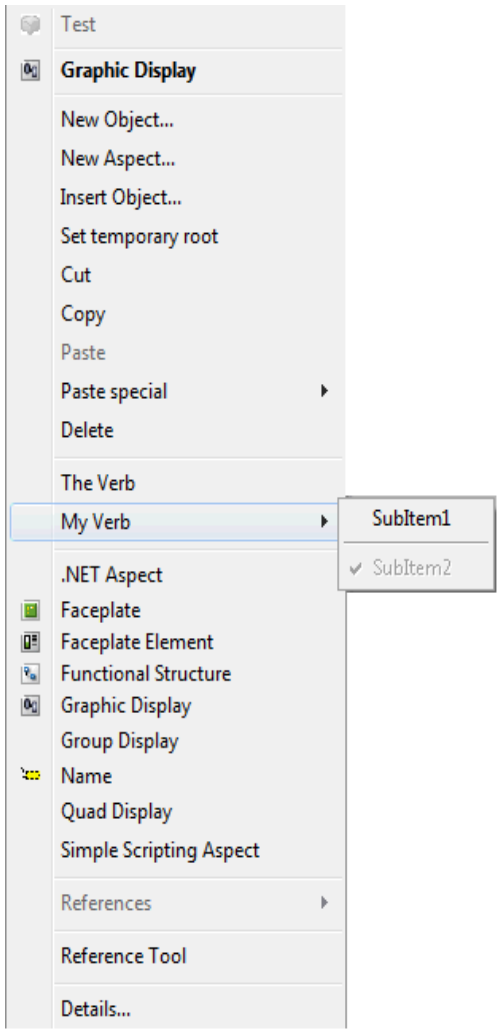

Figure 4. Context menu with two Submenu Items

Verbs that have been added in a verb controller do not have a specific verb method to run when the verb is invoked. Instead a verb dispatcher method should be

implemented for these verbs. The dispatcher is a method annotated with the ***AspectVerbDispatch*** attribute.

The dispatcher method should be ***static void*** and have two arguments, a verb site and a command id that identifies the verb being invoked.

```
[AspectVerbDispatch]
public static void VerbDispatch(IAspectVerbSite site, int cmdId)
{
    MessageBox.Show("Verb "+ cmdId.ToString());
}
```

The verb site has the following members

- `AspectRef Aspect { get; }`

  Specifies the identity of the aspect .

- `IBaseServices BaseServices { get; }`

  Denotes a reference to the base services that give access to Aspect and object-related operations.

- `T GetAspectData<T>();`

  Get aspect persistent data

- `Void SetAspectData<T>(T data);`

  Set aspect persistent data

- `TWorkplaceServices WorkplaceServices { get; }`

  Denotes a reference to 800xA workplace services

National Language Support

The names of aspect view and aspect verb can be handled automatically with National Language Support (NLS), using the standard Microsoft .NET internationalization. This user manual does not cover the entire internationalization process. It describes only the steps necessary to globalize an aspect implementation. For the localization process, see the Microsoft documents.

To ‘globalize’ an aspect implementation, all the user interface names should be replaced by the resource names. To create string resources for the names of aspect view and aspect verb, open the resource editor in Visual Studio and add the default view name and the verb name. These names will serve as fallback, if an appropriate language satellite assembly is missing. The process of producing satellite assemblies for different cultures is called ‘localization’, and will not be addressed here.

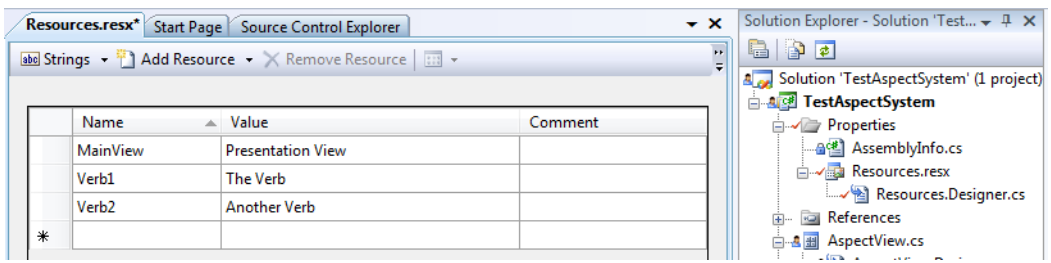

Figure 5. Adding Resources

Add the ***Resources*** attribute to the class implementing an aspect view or an aspect verb. The attribute specifies the base name for resource lookup within that class. The default name is ***<your namespace>*** followed by ***.Properties.Resources***. Verify this by looking in the ***Resource.Designer.cs*** file.

```
enum VerbID
{
    TheVerb,
    AnotherVerb
}
```

```
[Binding("{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}")]
[Resources("TestAspectSystem.Properties.Resources")]
public static class MyVerbs
{
    [AspectVerb(
        VerbID.TheVerb,
        "Verb1",
        ContextMenuContext.Object | ContextMenuContext.Aspect,
        UserRoleIndex.Operate)]
    public static void TheVerb(IAspectVerbSite site)
    {
    }
}
```

If the satellite resource assembly does not exist, then the verb in this example will be shown in the context menu as "The Verb", because this is the default resource.

System Resource Access

In Process Portal A, NLS-handled resources can be defined in resource aspects. These resources can then be read from program code using IResourceServices.

The API uses types suitable for WPF for resources. This means that WPF elements can be easily data-bound.

Using resources together with a WinForms control requires conversion to WinForms types. The following example shows how to set the background color on a WinForms control. The method ***ToDrawingColor*** converts the color to a ***System.Drawing.Color***.

```
System.Windows.Media.Color colour;
IResourceServices rs = this.ViewSite.BaseServices.ResourceServices;
colour = rs.GetLogicalColor("HighAlarmText").Colors.FirstOrDefault();
this.label2.BackColor = colour.ToDrawingColor();
```

The following example shows how to set the font on a WinForms control. Font info returned by the API has properties suitable for data binding to WPF controls. The method ***ToFont*** makes a conversion to ***System.Drawing.Font***.

```
FontInfo f = rs.GetFont("PPA9");
label2.Font = f.ToFont();
```

Aspect Default Activation

Default activation can be implemented by an aspect to specify the default action for the aspect. The default action is used by the context menu to mark the default menu item, and it is also used by the workplace to respond to double-click of the aspect. Default activation is either running an aspect verb or showing an aspect view.

If your aspect does not specify a default activation then Industrial IT system will decide what to do. Showing the main view is the default action of all aspects.

To set default activation a user control or verb method can be tagged with an ***AspectDefaultActivation*** attribute.

```
[AspectDefaultActivation]
```

In addition to applying this attribute the ***IAfwAspectDefaultActivation*** operation entry must be added to the implementation binding in the .add file. The operation is shown in bold face in the example below:

```
implBind
{
    id = "{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}"
    aspectId = "{7E4CD9F9-93FF-11D2-86B3-0000F87884BE}"
    name = Example
    clsid = "{1AAA2CB0-1EF1-4996-A987-B37604ADFEFC}"
    assembly = "TestAspectSystem.dll"
    operations {
           "{635D2C6F-1EA6-4D7B-9736-C011CB170C54}"
           "{9F01BF50-756E-11D3-8E17-0008C7A933F2}"
           "{C4B81CDA-3C4A-11D2-A1E0-0000F8788595}"
    }
}
```

System Access Examples

System access is illustrated through a number of use cases that are, more or less, realistic.

Use Case 1 - Finding the Local Computers Node Group

This example illustrates a way to find the name of the node group to which the local computer node belongs.

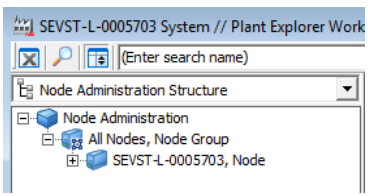

Figure 6. Node Administration Structure

```
using StructureCategories =
ABB.xA.Base.Constants.BasicAspects.AspectCategories.StructureCategories;
using BasicObjectTypes = ABB.xA.Base.Constants.BasicObjectTypes;

IAspectObjectsServices aos = site.BaseServices.AspectObjectsServices;
// Get all placements in the Node Administration structure of the
// local computer node object. Take the first one found and display
// the name of its parent object.
NodeId localNode = site.BaseServices.SecurityServices.GetLocalNode();
StructureNodeRef placement = aos.GetObjectPlacements(
    new ObjectId(localNode),
    StructureCategories.NodeAdministrationStructure).FirstOrDefault();
if (placement != null)
{
    StructureNodeRef parent = aos.GetParent(placement);
    ObjectInfo info = aos.GetObjectInfo(parent.ObjectId);
    MessageBox.Show(info.Name);
}
```

The output from this program is a dialog containing the text “All Nodes”.

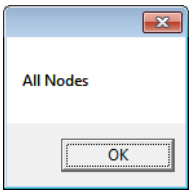

Figure 7. All Nodes

However, the method mentioned above has a problem. If the Node object has more than one placement in the Node Administration structure, the result is not reliable since an arbitrary placement is chosen.

It is obvious that more information is required to select the correct placements. The node may belong to multiple node groups. In the following example, we accept multiple parents of the object type ***Node Group***.

```
// Get all placements in the Node Administration structure of the
// local computer node
NodeId localNode = site.BaseServices.SecurityServices.GetLocalNode();
IEnumerable<StructureNodeRef> placements = aos.GetObjectPlacements(
    new ObjectId(localNode),
    StructureCategories.NodeAdministrationStructure);
// Find the parents that are Node Group objects
foreach (var placement in placements)
{
    StructureNodeRef parent = aos.GetParent(placement);
    if (aos.GetObjectType(parent.ObjectId) == BasicObjectTypes.NodeGroup)
    {
        ObjectInfo info = aos.GetObjectInfo(parent.ObjectId);
        MessageBox.Show(info.Name);
    }
}
```

Use Case 2 - Looking up Object Placements

This example shows how to find all the placements in the Functional Structure of all the objects named "Root". For this, we use a lookup function that gets its information from Structure and Name Server.

```
using StructureCategories =
ABB.xA.Base.Constants.BasicAspects.AspectCategories.StructureCategories;
using NameCategories =
ABB.xA.Base.Constants.BasicAspects.AspectCategories.NameCategories;

IAspectObjectsServices aos = site.BaseServices.AspectObjectsServices;

// Only consider basic name
AspectCategoryId[] names = {NameCategories.Name};
// Only consider Functional Structure
AspectCategoryId[] structs = { StructureCategories.FunctionalStructure };

IEnumerable<StructureNodeRef> placements =
    aos.LookupObjectsPlacements("Root", names, structs, false);
```

If the structure restrictions are removed, and placements are accepted in all the structures, then it becomes more compact. If no ***name*** and no ***structure*** categories are specified, then all structures and only names of the category ***Name***, are included in the search scope.

```
IEnumerable<StructureNodeRef> placements =
    aos.LookupObjectsPlacements("Root");
```

Section 4 Integration

Definition Files (.add)

In System 800xA, aspect implementations are added to the system through ***.add*** files. The ***.add*** files contain information about Aspect System, Aspect Types, Aspect Categories, Implementation Bindings, and Operations.

The easiest way to create and maintain ***.add*** files is to use ***AfwAdminTool*** 

located in the ***bin*** folder of Process Portal A SDK.

Specifications of .NET implementations are not supported directly by ***AfwAdminTool***. Instead, they have to be entered into the ***.add*** file using a text editor.

Table 2 provides an overview of the elements in an ***.add*** file.

Table 2. Overview of the Elements in an .add File

Field	Description
Aspect System	An aspect system is a container for a collection of aspect types. Its purpose is purely administrative.
Aspect Type	An aspect type carries the complete implementation of an aspect.

Table 2. Overview of the Elements in an .add File (Continued)

Field	Description
Aspect Category	In System 800xA, an aspect instance is created from an aspect category. An aspect category represents a 'type', in object-oriented terms. An aspect category is a subtype of an aspect type. Aspect categories derived from the same Aspect Type only differ by identity. It is the aspect type that carries all the implementation.
Implementation Binding	The implementation of an aspect type is specified through the implementation bindings tied to the aspect type. The implementation binding can specify operations to identify supported interfaces. It also specifies a COM object that implements the interfaces.

For more details on implementing an aspect using C#, see Section 5, Tutorial.

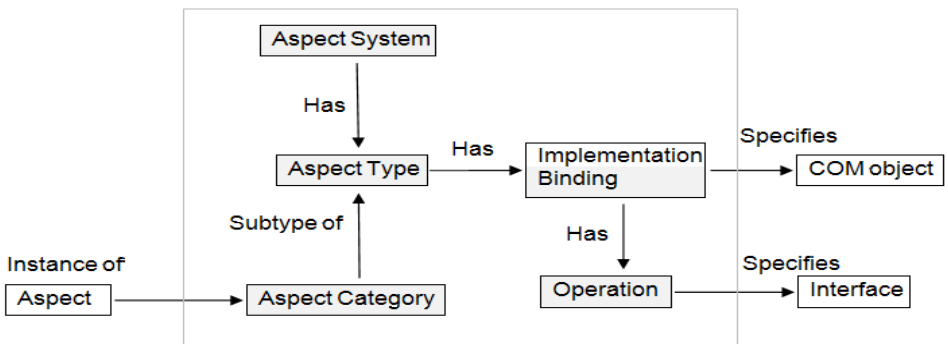

Figure 8. Elements of an .add file

Definition Files and .NET Assemblies

When an aspect is implemented in .NET, the ***.add*** file should be edited using a plain text editor like Notepad, because ***AfwAdminTool.exe*** does not support .NET implementation bindings.

The following example is an implementation binding entry, where the implementation is realized in .NET.

```
implBind
{
    id = "{7E4CD9F8-93FF-11D2-86B3-0000F87884BE}"
    aspectId = "{7E4CD9F9-93FF-11D2-86B3-0000F87884BE}"
    name = Example
    clsid = "{1AAA2CB0-1EF1-4996-A987-B37604ADFEFC}"
    assembly = "{9026671F-E2BE-4C15-BF7A-3F37DE6EF741}:Tutorial.dll"
    operations {
        "{635D2C6F-1EA6-4D7B-9736-C011CB170C54}"
    }
}
```

The ***id*** and ***aspectId*** entries should be unique GUIDs created with an ID-generator. One such entry is available under the **Tools** menu in Visual Studio.

The ***clsid*** must always be `"{1AAA2CB0-1EF1-4996-A987-B37604ADFEFC}"`. This is the ***clsid*** of a COM object in Process Portal A that acts as an adapter for the .NET assembly. See Figure 1.

The implementation binding example has a single operation specified under Operations. The operation identifier in the example corresponds to ***IAfwAspectViewControl***. This operation specifies that the binding implement an aspect view.

A new value ***assembly*** is introduced to specify the assembly ***.dll*** file name.

The name of the ***.dll*** file should be written within quotation marks.

The assembly name should include the system extension ID, indicating where the assembly is located. If the system extension ID is missing, the system searches for the assembly in the ***bin*** folder of Process Portal A. This is convenient for prototyping, but for a real product, a separate installation folder should be specified.

In System 800xA, functionality is added through system extensions. The developer may or may not have a released product in the form of a system extension for PPA.

If the developer has a released product and its system extension, then follow these steps:

1. Put the new implementation binding specification entries in the existing ***.add*** file.
2. Take the system extension ID from the system extension of the existing product and place it in front of the assembly name in the existing ***.add*** file.

   For the syntax, see the example below:

```
assembly = "{9026671F-E2BE-4C15-BF7A-3F37DE6EF741}:Tutorial.dll"
```

However, if the developer does not have a released product and its system extension, the ***.add*** file is also **not** available. In this case, to associate a new aspect with a **new** product, follow these steps:

1. Write an ***.add*** file containing a description of the functionality offered by the aspect. Use the ***AfwAdminTool*** located in the ***bin*** folder of Process Portal A SDK to write the ***.add*** file.
2. Take the system extension ID from the system extension of the new product.
3. Put the implementation binding specification entry in the newly created ***.add*** file using a plain text editor like Notepad, because ***AfwAdminTool.exe*** does not support .NET implementation bindings.
4. Place the system extension ID in front of the assembly name in the new ***.add*** file.

The new aspect will now be associated with the new product.

Once a system extension is loaded into the Process Portal A, the system will be able to locate the assembly. The assembly must be placed in the ***bin*** folder of the system extension.

A system extension is created using Aspect Studio or Aspect Express. The following lines are taken from the definition file of Aspect Express Packager, where the value `SysExtGUID` is the system extension ID.

```
<Product>
<Name>Demo</Name>
<FileVersion>1.0</FileVersion>
<FileInfo>Aspect Express product configuration file</FileInfo>
<ProductGUID>7CD41765-77DB-4E6B-BBD2-8DA3A9D3A7A7</ProductGUID>
```

```
<UpgradeGUID>4371C5FF-516F-4A69-B6A0-DB8F841AA0EA</UpgradeGUID>
<SysExtGUID>9026671F-E2BE-4C15-BF7A-3F37DE6EF741</SysExtGUID>
```

If Aspect Studio is used for packaging of the product, the system extension is defined by a C++ COM project.

The following example is an excerpt from the Process Portal A system extension project where ***CLSID_AfwAIPSysExt*** is the system extension ID.

```
class ATL_NO_VTABLE CProcessPortal :
    public CComCoClass<CProcessPortal, &CLSID_AfwAIPSysExt>,
    public IAfwSystemExtensionInfo,
    public CExtensionHelper,
    public IAfwSystemExtensionUpgrade,
    public IAfwCreateSystem
{
public:
    CProcessPortal() :
    m_nServerType(0),
    m_dwCBCookie(0),
    m_pCreator(NULL),
    m_bAddPropertyServer (false),
    CExtensionHelper(
        L"Aspect_Integrator_Platform",
        NULL,
        &CLSID_AfwAIPSysExt,
        IDS_NAME,
        IDS_DESCRIPTION,
        0,
        0,
        0)
    {
        m_version = _AFW_VERSION;
        m_buildNumber = _AFW_BUILD_NO;
    }
    …
};
```

The ***.idl*** file contains the GUID that is recognized as ***CLSID_AfwAIPSysExt*** system extension ID. The line `assembly = "{E4D66305-21F0-4397-8CE7-CF06736591BD}:Tutorial.dll"` specified in the implementation binding in the ***.add*** file indicates that the assembly file ***Tutorial.dll*** is located in the ***Process Portal A \ bin*** directory. The ***midl*** compiler creates the ***CLSID_AfwAIPSysExt*** system extension ID from `uuid(E4D66305-21F0-4397-8CE7-CF06736591BD)` in the ***.idl*** file excerpt below.

```
library AFWAIPVERSIONLib
{
    importlib("stdole32.tlb");
    importlib("stdole2.tlb");
    importlib("AfwAbsInterfaces.dll");
    importlib("AfwAohInterfaces.dll");

    [
        uuid(E4D66305-21F0-4397-8CE7-CF06736591BD),
        helpstring("AfwAIPSysExt Class")
    ]
    coclass AfwAIPSysExt
    {
        [default] interface IUnknown;
    };
};
```

Section 5 Tutorial

This tutorial describes how to create .NET aspect views using C# and how to work with persistent data. An aspect view can be built as a WinForms or WPF control. This tutorial describes how to create an aspect view with WinForms control.

Preparations

Creating the C# Project

Create a new Windows Forms Control Library project and name it ***Tutorial***. The resulting assembly file now gets the name ***Tutorial.dll*** and the default namespace will be ***Tutorial***.

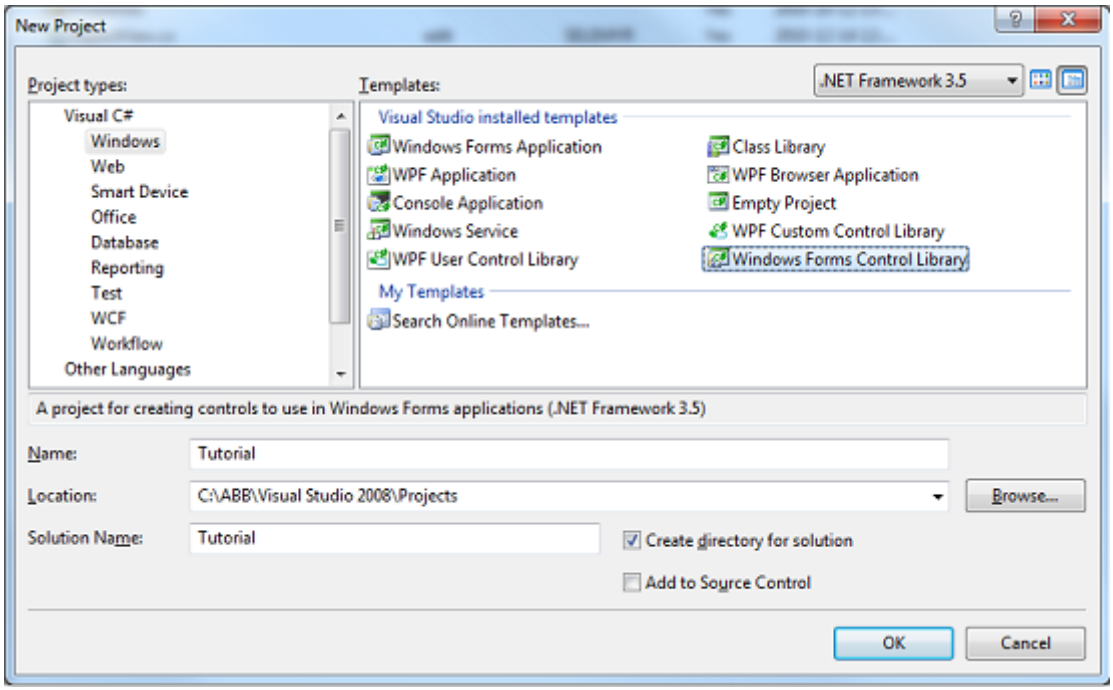

Figure 9. New Project Dialog in Visual Studio

Implementation Binding

Before continuing with the user controls, add an implementation binding to the ***.add*** file. For details on the entries of an implementation binding, see the topic Definition Files and .NET Assemblies on page 48.

In many cases, an ***.add*** file may be already there as aspects are already written in C++/COM. If there is an ***.add*** file and the new aspects should be a part of the same product, then add the new contents to the existing file.

Add a new implementation binding to the ***.add*** file. This binding will represent the new aspect view.

Create a new ***ID*** and a new ***Aspect ID*** using the Create GUID tool in Visual Studio. Select **Registry Format**, as shown in Figure 10.

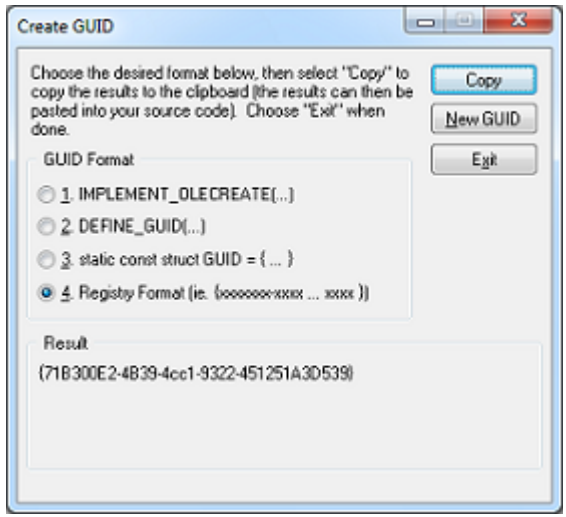

Figure 10. GUID Tool in Visual Studio

```
implBind
{
    id = " {EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}"
    aspectId = "{C04BF975-B2DE-4464-805B-B31B8FED5056}"
    name = "TutorialViews"
}
```

Add the ***clsid*** entry. The value for this ID is the same for all implementation bindings that refers to a .NET assembly. See the topic Definition Files and .NET Assemblies on page 48 for details.

```
clsid = "{1AAA2CB0-1EF1-4996-A987-B37604ADFEFC}"
```

Add the ***assembly*** entry:

```
assembly = "{9026671F-E2BE-4C15-BF7A-3F37DE6EF741}:Tutorial.dll"
```

The assembly value should be the name of the assembly file. In this case, the name of the assembly file should be ***Tutorial.dll*** with the system extension ID of the

product added at the front of the name. See the topic Definition Files and .NET Assemblies on page 48 for a detailed discussion on ***.add*** files.

Finally, add the operation `IAfwAspectViewControl` to the implementation binding. The GUID for the operation is found in the file ***AfwWebAsoSupport.add***.

This operation informs the system that the implementation binding is specifying aspect views through this interface.

Now we have a complete implementation binding for the aspect view.

```
implBind
{
    id = " {EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}"
    aspectId = "{C04BF975-B2DE-4464-805B-B31B8FED5056}"
    name = "TutorialViews"
    clsid = "{1AAA2CB0-1EF1-4996-A987-B37604ADFEFC}"
    assembly = "{9026671F-E2BE-4C15-BF7A-3F37DE6EF741}:Tutorial.dll"
    operations {
        "{635D2C6F-1EA6-4D7B-9736-C011CB170C54}"
    }
}
```

Implementing the Main View

To become an aspect view, a user control needs to implement the ***IAspectView*** interface from the ***ABB.xA.Base.Development*** namespace in the ***ABB.xA.Base.dll*** assembly. Add a reference to the ***ABB.xA.Base*** assembly.

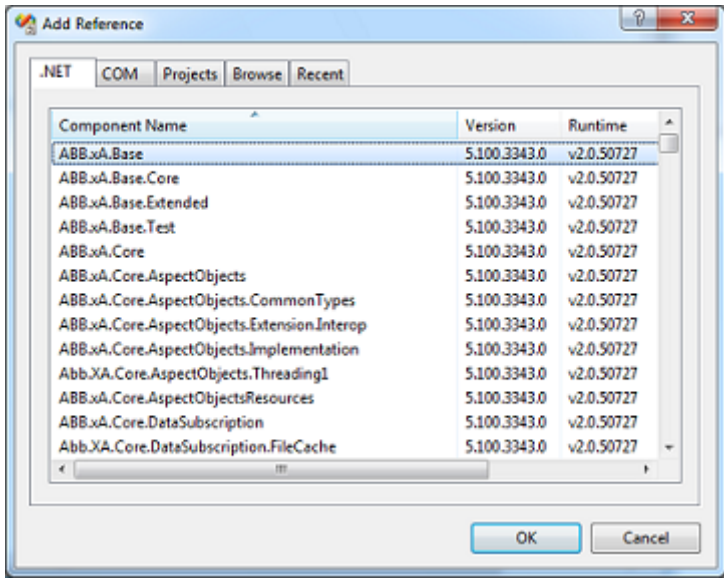

Figure 11. Add Reference Dialog

Add two ***using*** statements.

```
using ABB.xA.Base;
using ABB.xA.Base.Development;
```

To implement ***IAspectView***.

```
#region IAspectView Members

public void Connecting()
{
}
public void Disconnecting()
{
}
```

```
public IAspectViewSite ViewSite { get; set; }

#endregion
```

The user control also needs a couple of attributes. The ***AspectView*** attribute tells the system that this class is an aspect view and attaches some configuration information to it. In the following example, the attribute tells the system that this view is the main view.

```
[AspectView(ViewId.Main, "Main view", UserRoleIndex.View)]
```

The ***Binding*** attribute informs the system about the implementation binding to which this class belongs. The attribute carries a GUID that should match the implementation binding ID in the ***.add*** file for the aspect system.

```
[Binding("{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}")]
```

The following example shows the minimum implementation required for an aspect view realized in .NET.

```
namespace Tutorial
{
    using System.Windows.Forms;
    using ABB.xA.Base;
    using ABB.xA.Base.Development;
    [Binding("{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}")]
    [AspectView(ViewId.Main, "Main view", UserRoleIndex.View)]
    public partial class MainView: UserControl, IAspectView
    {
        #region Construction
        public MainView()
        {
           this.InitializeComponent();
        }
        #endregion
        #region IAspectView Members
```

```
            public void Connecting()
            {
            }
            public void Disconnecting()
            {
            }
            public IAspectViewSite ViewSite { get; set; }
            #endregion
        }
}
```

Implementing a Config View

In this tutorial, we will create a new aspect view configured as a configuration view. Use the standard ID ***ViewId.ConfigView*** and the same implementation binding ID as the main view.

```
[Binding("{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}")]
[AspectView(ViewId.Config , "Config view",
    UserRoleIndex.ConfigView)]
public partial class ConfigView: UserControl, IAspectView
. . .
```

Persistent Data

Persistent data can be added to the aspect views.

Configuration data is stored persistently in the aspect blob using WCF (Windows Communication Foundation) technology. The data is serialized and deserialized using WCF serialization.

Add a reference to the ***System.Runtime.Serialization*** assembly to work with WCF data contracts.

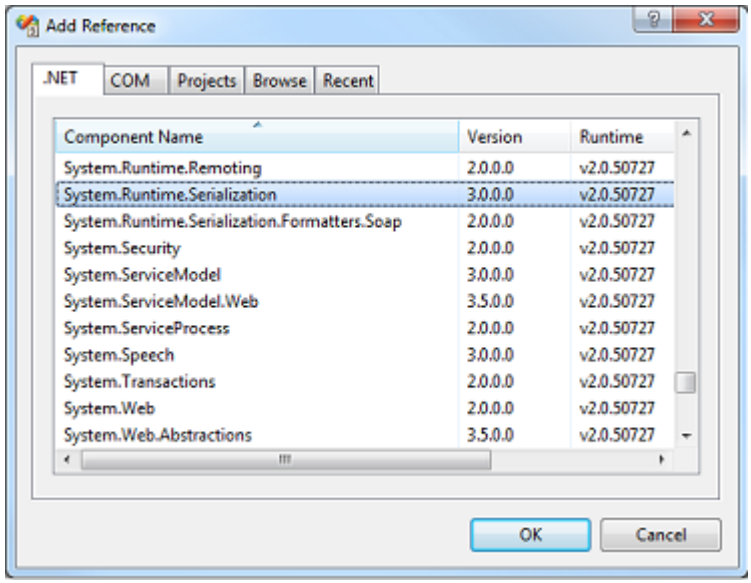

Figure 12. Add Reference Dialog

Create a new class and name it ***ConfigData***. Add the ***DataContract*** attribute and the ***using*** statements listed below. Add a new member of type ***string*** and annotate it with the ***DataMember*** attribute.

```
using System.Runtime.Serialization;
using ABB.xA.Base;
[DataContract(Name = "Data")]
public class ConfigData
{
    [DataMember]
    public string Name { get; set; }
}
```

Config View

Create a new user control named ***ConfigView*** and add the attributes, as shown in the following example. Add a textbox named ***txtName*** and a button named ***btnApply***. Now, when the user clicks the **Apply** button, the text in the text box will be stored persistently.

```
namespace Tutorial
{
    using System.Windows.Forms;
    using ABB.xA.Base;
    using ABB.xA.Base.Development;

    [Binding("{EB9D0CAE-9F1B-4c93-AB79-C99E86DBA2D4}")]
    [AspectView(ViewId.ConfigView, "Config view",
        UserRoleIndex.ConfigView)]
    public partial class ConfigView: UserControl, IAspectView
    {
        #region Private data
        private ConfigData configData;
        #endregion
        #region Construction
        public ConfigView()
        {
           this.InitializeComponent();
        }
        #endregion
        #region IAspectView Members
        public void Connecting()
        {
           this.UpdateUI();
        }
        public void Disconnecting()
        {
        }
        public IAspectViewSite ViewSite { get; set; }
        #endregion
```

```
        #region Event handlers
        private void btnApply_Click(object sender, System.EventArgs e)
        {
           this.configData.Name = this.txtName.Text;
           this.ViewSite.SetAspectData<ConfigData>(this.
           configData);
        }
        #endregion
        #region Private methods
        private void UpdateUI()
        {
           this.configData = this.ViewSite.GetAspectData<ConfigData>();
           this.txtName.Text = this.configData.Name;
        }
        #endregion
    }
}
```

Reacting to Events

Subscribe to the ***AspectChanged*** event that is available on the ***ViewSite*** to act when the aspect data is changed. Change the implementation of the ***Connecting*** method to the following.

```
public void Connecting()
{
    this.ViewSite.AspectChanged += (sender , e) =>
    {
        // We are only interested in persistent data changes
        if ((e.Details & AspectChangeDetails.Blob) != 0)
        {
           this.UpdateUI();
        }
    };
    this.UpdateUI();
}
```

This code sets up an event handler, using a ***lambda expression*** that calls the ***UpdateUI*** method when the aspect blob is changed.

To write the code in an orthodox manner, see the following example.

```
public void Connecting()
{
    this.ViewSite.AspectChanged += new
    EventHandler<AspectChangedEventArgs>(ViewSite_AspectChanged);
    this.UpdateUI();
}

void ViewSite_AspectChanged(object sender, AspectChangedEventArgs e)
{
    // We are only interested in persistent data changes
    if ((e.Details & AspectChangeDetails.Blob) != 0)
    this.UpdateUI();
}
```

Add a private Boolean member named ***modified*** to be used for indicating that data has been modified locally. Add the ***text changed*** event to the textbox.

```
private void txtName_TextChanged(object sender, System.EventArgs e)
{
    this.modified = true;
    this.btnApply.Enabled = true;
}
```

Add a ***SaveData*** method.

```
private void SaveData()
{
    this.configData.Name = this.txtName.Text;
    this.ViewSite.SetAspectData<ConfigData>(this.configData);
    this.modified = false;
}
```

Modify the ***btnApply_Click*** method and the ***UpdateUI*** method.

```
private void btnApply_Click(object sender, System.EventArgs e)
{
    this.SaveData();
}
private void UpdateUI()
{
    this.configData = this.ViewSite.GetAspectData<ConfigData>();
    this.txtName.Text = this.configData.Name;
    this.modified = false;
    this.btnApply.Enabled = false;
}
```

Change the implementation of the ***Disconnecting*** method to the following.

```
public void Disconnecting()
{
    if (this.modified == true)
    {
        DialogResult dialogResult = MessageBox.Show("The configuration
        data has been modified.\r\n
        Do you want to save the modified data?", "Save configuration
        data?", MessageBoxButtons.OKCancel);
        if (dialogResult == DialogResult.OK)
           this.SaveData();
    }
}
```

Data Versioning

For a released product, a new data member has to be added to the configuration data class, sooner or later. However, there are now aspects that have their data saved in the aspect blob in the old data format. To handle this, a data versioning scheme is required where old data can be unserialized into a new version of the data class, and WCF has a built-in support for this.

As described in this tutorial, to add an object ID field to the data, add a new class named ***ConfigData2***, in the same way as how the ***ConfigData*** class was implemented.

```
[DataContract(Name = "Data")]
public class ConfigData
{
    [DataMember]
    public string Name { get; set; }
}
```

In the ***ConfigData2*** class, add an ***ObjectId*** property. To set default values, use the ***OnDeserializing*** attribute and implement a method called ***OnDeserializing*** that sets the default values before the deserialization takes place. If the ***Name*** or ***ObjectId*** properties are set in the data, the default values will be overridden.

```
[DataContract(Name = "Data")]
public class ConfigData2
{
    [DataMember]
    public string Name { get; set; }

    [DataMember]
    public ObjectId ObjectId { get; set; }

    [OnDeserializing]
    void OnDeserializing(StreamingContext context)
    {
        this.Name = "Default text";
        this.ObjectId = ObjectId.Null;
    }
}
```

Now, change the implementation so that the ***ConfigData2*** type is used instead of the ***ConfigData*** type. Also, call the ***GetAspectData*** and the ***SetAspectData*** methods with ***ConfigData2*** as the type ***parameter***. These changes have been applied in the code sample in the next topic System Access on page 66.

System Access

The system can be accessed using methods on ***ViewSite.BaseServices***. In the following example, the ***UpdateUI*** method has been added to the ***MainView*** class. The code reads an object ID from the data. If the object ID is null, look up an object with the name stored in the aspect data. Use the ***FirstOrDefault*** extension method provided by the ***System.Linq*** namespace to get the first object ID, or else use ***null***, if the object does not exist. If there is an object ID, use the ***GetObjectInfo*** method to get information about the object.

```
public void UpdateUI()
{
    this.configData = this.ViewSite.GetAspectData<ConfigData2>();

    // Get the object id from the configuration data. If the the object id
    // is null, then we lookup the object by name
    var objectId = this.configData.ObjectId;
    if (objectId == null || objectId == ObjectId.Null)
    {
        var objectName = this.configData.Name;
        var objects = this.ViewSite.BaseServices.
           AspectObjectsServices.
           LookupObjects(objectName);
        objectId = objects.FirstOrDefault();
    }
    if (objectId != null)
    {
        var objectInfo = this.ViewSite.BaseServices.
           AspectObjectsServices.GetObjectInfo(objectId);
    }
    // TODO: Use objectInfo to update UI elements
}
```

Sending Audit Messages

In this example, a new system message is created that will be used to send an audit message when the ***Name*** property is changed in the ***aspects*** data.

System message definitions can be created using the Message Definition Tool (***AfwSysMsgDefineTool.exe***).

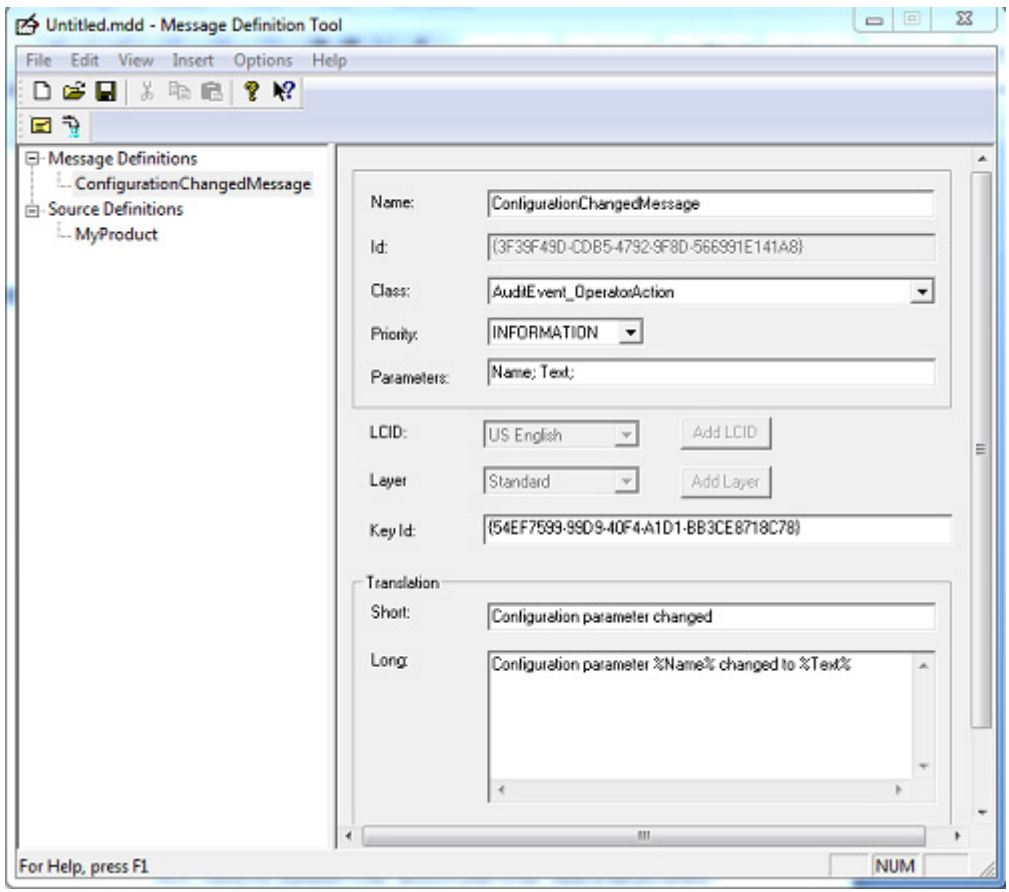

Figure 13. Message Definition Tool

To create system message definitions, follow these steps:

1. Open this ***.exe*** file located in the ***..\Process Portal A\bin*** directory.
2. Save the system message as ***Tutorial.mdd***.
3. Go to the ***Insert*** menu option and select ***Message Definition***. Change the default name from ***MessageDefinition-1*** to ***NameChange***. Change the class to ***AuditEvent_ConfigurationChange*** and add ***FromValue; ToValue***; in the parameters field. Add the text "***Name changed from '%FromValue%' to '%ToValue%'***" in the ***Short*** field of the ***Translation*** group.

The resulting message should look like the following example, except for the ***ID*** field. The ID field is system-generated, so that your ID will be different.

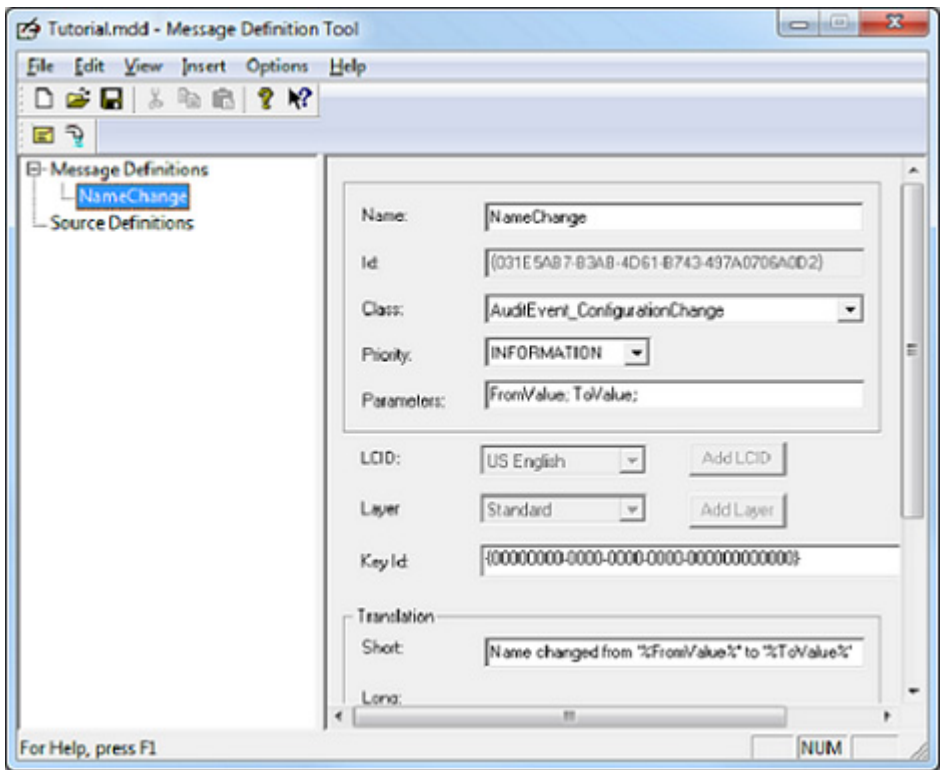

Figure 14. Adding Message Definition

4. Go to the ***Insert*** menu and select ***Source Definition***. Change the default name from ***SourceDefinition-1*** to ***Tutorial***. Add the text "***Tutorial message definition***" in the ***Short*** field of the ***Translation*** group.

The resulting source definition should look like the following example, except for the ***ID*** field.

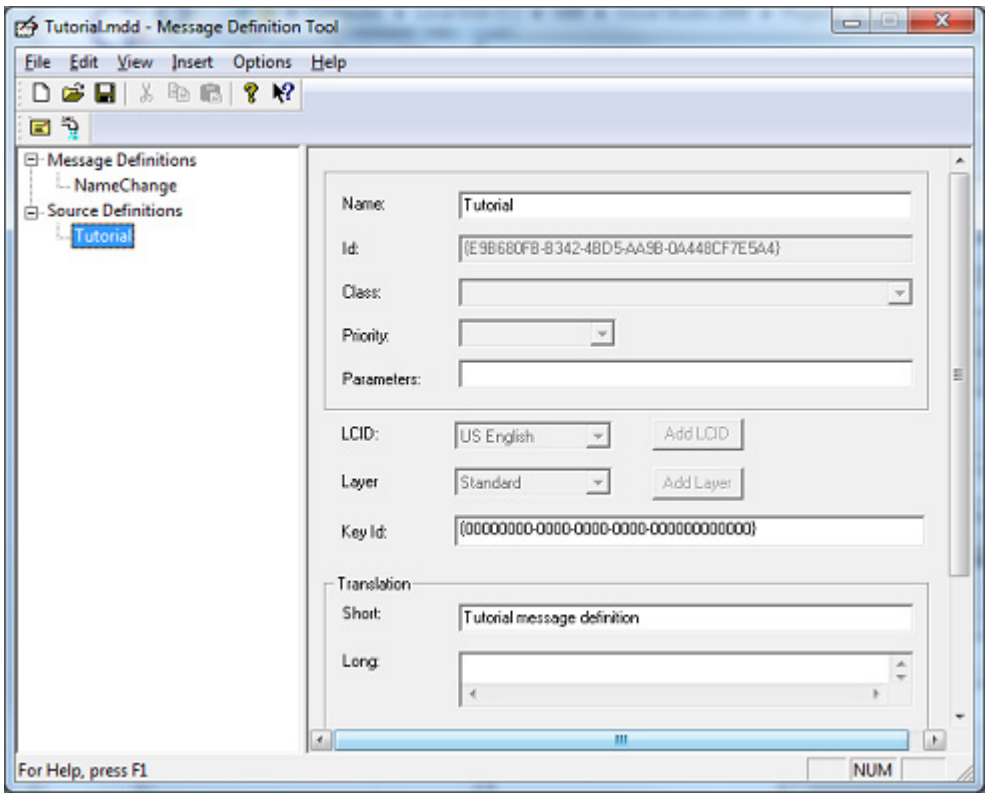

Figure 15. Adding Source Definition

Add the system message to the system by selecting ***Register Definitions*** … in the ***File*** menu.

Now, using the newly created message, send an audit message using the ***SecurityServices.GenerateAuditEvent*** method. In the following example, the implementation of the ***SaveData*** method has been changed to generate an audit event. The ***MessageId*** should be the value found in the ***ID*** field of the ***Tutorial*** source definition, as shown in Figure 15. The value in the ID field of the ***NameChange*** message definition should be ***MessageSourceId***, as shown in Figure 14.

```
private void SaveData()
{
    // Send audit message
    this.ViewSite.BaseServices.SecurityServices.GenerateAuditEvent(
        this.ViewSite.Aspect,
        AuditTransactionType.ModifyConfiguration,
        true, // Postpone until transaction completed
        new MessageSourceId("{E9B680FB-B342-4bd5-AA9B-0A448CF7E5A4}"),
        new MessageId("{031E5AB7-B3AB-4d61-B743-497A0706A0D2}"),
        new SystemMessageParameter("FromValue", this.configData.Name ??
        ""),
        new SystemMessageParameter("ToValue", this.txtName.Text));

    // Save aspect data
    this.configData.Name = this.txtName.Text;
    this.ViewSite.SetAspectData<ConfigData2>(this.configData);
}
```

Revision History

Introduction

This section provides information on the revision history of this User Manual.

The revision index of this User Manual is not related to the 800xA 5.1 System Revision.

Revision History

The following table lists the revision history of this User Manual.

Revision Index	Description	Date
-	Created for 800xA 5.1 Feature Pack 1	August 2011
A	Updated for 800xA 5.1 Feature Pack 2	December 2011

Updates in Revision Index A

The following table shows the updates made in this User Manual for 800xA 5.1 Feature Pack 2.

Updated Section/Sub-section	Description of Update
Section 4 : Programming	Following topics are added: Introduction Aspect Views Aspect Default Activation Following topics are updated: Aspect Verbs Headings "Aspect Verbs" and "System Messages" are changed to Heading2 Chapter "View Site" removed. Contents moved to the new chapter "Aspect Views”
Section 5 : Tutorial	Removed an example on page 48
Revision History	New section is added

Contact us

ABB AB
Control Technologies
Västerås, Sweden
Phone: +46 (0) 21 32 50 00
Fax: +46 (0) 21 13 78 45
E-Mail: processautomation@se.abb.com

www.abb.com/controlsystems

ABB Inc.
Control Technologies
Wickliffe, Ohio, USA
Phone: +1 440 585 8500
Fax: +1 440 585 8756
E-Mail: industrialitsolutions@us.abb.com

www.abb.com/controlsystems

ABB Pte Ltd
Control Technologies
Singapore
Phone: +65 6776 5711
Fax: +65 6778 0222
E-Mail: processautomation@sg.abb.com

www.abb.com/controlsystems

ABB Automation GmbH
Control Technologies
Mannheim, Germany
Phone: +49 1805 26 67 76
Fax: +49 1805 77 63 29
E-Mail: marketing.control-products@de.abb.com

www.abb.de/controlsystems

ABB Automation LLC
Control Technologies
Abu Dhabi, United Arab Emirates
Phone: +971 (0) 2 4938 000
Fax: +971 (0) 2 5570 145
E-Mail: processautomation@ae.abb.com

www.abb.com/controlsystems

2PAA107043-510 A

Power and productivity
for a better world™