Business Controls Development

DELMIA Apriso 2021 Technical Guide

3DEXPERIENCE®

DASSAULT SYSTEMES | The 3DEXPERIENCE® Company

Contents

Figures

1 Introduction

1.1 Objective

The objective of this document is to introduce programmers to the concept of developing reusable Function Interpreter visual Web components called Business Controls. The development process will be standardized through the introduction of a set of guidelines that should be followed by programmers.

1.2 Scope

The scope of this document describes the following core areas of Business Control development:

- General information about the development framework (2 Framework Overview)
- Components required for development of Business Controls (3 Requirements)
- Process of Business Controls creation (4 Business Control Creation Process):
- Best practices (5 Best Practices)

1.3 Definitions

Business Control – a set of classes and ASP.NET web controls that allow for embedding pre-coded Web forms and related business logic into Function Interpreter Processes in a simple and easily configurable way.

2 Framework Overview

2.1 Motivation

Some of the Functions configured in DELMIA Apriso Process Builder appear in many Processes and Operations, often with few or no changes in appearance and behavior. It is not always possible to reuse some existing Functions, because they are defined to work in specific contexts and are not defined to be reusable or customizable in the future.

With that in mind, Business Controls have been introduced. The Business Control technology enables developing UI components that can be placed in Process Builder Operations as Functions with many configuration features for modifying the component's outlook and behavior. These are later displayed by Function Interpreter in the desired fashion.

Business Controls are templates for Functions and are therefore reusable components. Business Controls can be designed to work with many different sets of Inputs, which allows them to speed up the development of production Operations and assure the consistent display of similar Functions to the end user.

2.2 Assumptions

All of the controls are implemented using the Microsoft® .NET framework and ASP.NET technology. Though the majority of them are written in the C# language, support for other .NET languages is also provided.

2.3 Business Control Architecture

A Business Control is a composite component that consists of several subcomponents. All of these subcomponents have to be developed according to the rules specified in the following sections of this document in order to assure proper and consistent behavior.

The main components of a Business Control are:

- **Business Control Class**

  The class is the axis of the entire Business Control, as it indicates what components should be used in the process of configuration and execution of the control. It implements the IBusinessControl interface (which is a member of `FlexNet.FunctionInterpreter.BusinessRules.Functions`) and is marked with the *ComponentRepositoryComponentType* and *BusinessControl* attributes.

- **Win Business Control Configuration**

  This is the WinForms user control. This control is used by Process Builder in design mode to set the Business Control properties.

- **Business Control Runtime**

  This is a set of ASP.NET user Web controls (Desktop and Mobile) which are used by Function Interpreter during execution to display the Business Control based on the properties set in Process Builder.

- **Business Control Properties**

  This is an optional class that contains the configuration options of the Business Control. It is persisted in the database so the class itself and all of its fields have to be serializable (or marked with the *XmlIgnore* attribute).

  It is recommended to implement the INotifyPropertyChanged interface on this class, because this is very useful during the writing of the Win Configurator.

3 Requirements

This chapter describes the tools and definitions required for development of Business Controls.

3.1 Tools

3.1.1 SDK

To develop and extend the DELMIA Apriso Business Controls, Microsoft® .NET SDK have to be installed. This library can be freely downloaded and installed from the Microsoft® Web site.

3.1.2 Programming Language

Most DELMIA Apriso Business Controls are written in the C# programming language. Business Controls can be extended in any language that is available for the .NET platform.

3.1.3 Development Environment

Microsoft® Visual Studio offers the best development environment for the .NET platform and is recommended as the primary developer tool.

3.2 Definitions

3.2.1 Interfaces

The interfaces used by Business Control runtime controls are defined in `FlexNet.FunctionInterpreter.BusinessRules.Functions.dll`.

The interfaces required to create the WinForms configurator for DELMIA Apriso Process Builder are defined in `FlexNet.ProcessBuilder2.BusinessRules.dll`.

3.2.2 Base Class

The base Business Control class is defined in `FlexNet.WebUI.BusinessControls.dll`.

3.2.3 Common Controls and Actions

Common controls and actions useful in development (e.g., creating/deleting Inputs/Outputs, changing Input/Outputs types, changing the routing) are defined in `FlexNet.ProcessBuilder2.WinUI.dll`.

4 Business Control Creation Process

To create a Business Control, create a new solution in the 3.1.3 Development Environment and add three new projects:

- **Business Control Properties**

  This should be a class library project which will contain the Business Control Properties Class for the configurator and also for the runtime controls. This assembly should reference as few DLLs as required, because it will be shared between the client and the server application and the Properties class will be sent via remoting services. If you already have properties for your old Web Business Control, you can use this class but you have to implement a few interfaces on it.

  For details on this project, refer to 4.1 Developing the Business Control Properties Project.

- **Business Control Runtime**

  This should be the Web Application project which will contain the main Business Control class and all the runtime classes.

  For details on this project, refer to 4.2 Developing the Business Control Runtime Project.

- **Business Control Win Editor**

  This should be the class library project which will contain the set of classes for the Business Control WinForms Editor for Process Builder. If your control does not need to have a WinForm Editor (e.g., it just has some predefined Inputs), you do not need to create this assembly.

  For details on this project, refer to 4.3 Developing the Business Control Win Editor Project.

  All these DLLs have to be signed with a strong name, because they will be placed in the .NET Global Assembly Cache or downloaded via ClickOnce as described in 4.4 Using a Business Control in DELMIA Apriso.

4.1 Developing the Business Control Properties Project

To create the project, perform the following tasks:

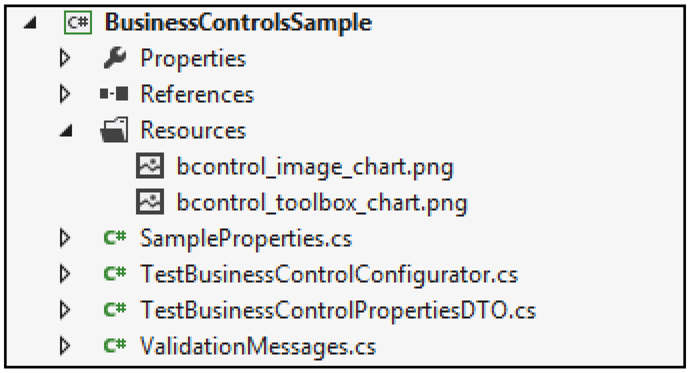
1. Create a Class Library project. The example project is named **BusinessControlsSample**.
2. Add the following classes to the folder:
   a. Business Control Configurator class. For instructions on creating this class, refer to 4.1.1 Developing the Business Control Win Configurator Class.
   b. Business Control Properties class – in the simplest case, you do not need to create a Properties class, as it could be a simple type such as *integer*, *string*, or an *array* of one of these types. Additionally, it could be a PropertyBag class. In the example solution described in this document, you can see the SampleProperties class and

TestBusinessControlPropertiesDTO class, which is used in the Process Builder solution. If your implementation is not complex, it may be sufficient to implement the IBusinessControlProperties interface on the SampleProperties class instead of creating a separate TestBusinessControlPropertiesDTO class.

For instructions on creating the Business Control Properties class, refer to 4.1.2 Developing the Business Control Win Properties Class.

3. Add references for:
   - `FlexNet.ProcessBuilder2.BusinessRules.dll` this assembly contains all required interfaces for the Win Configurator control)
   - `System.Drawing.dll`
   - `System.Web.dll`
   - `FlexNet.SystemServices.dll`
   - `FlexNet.DesignStudio.SystemServices.dll`
   - `FlexNet.DataServices.Process.Data`
   - `FlexNet.FunctionInterpreter.Systemservices.Transport.dll`
   - `FlexNet.BusinessRules.BusinessControls2.dll`

The project tree should look similar to the one shown in the figure below.



```
BusinessControlsSample
    Properties
    References
    Resources
        bcontrol_image_chart.png
        bcontrol_toolbox_chart.png
    SampleProperties.cs
    TestBusinessControlConfigurator.cs
    TestBusinessControlPropertiesDTO.cs
    ValidationMessages.cs
```

Figure 1 Sample Business Control properties project tree

4.1.1 Developing the Business Control Win Configurator Class

The Business Control Win Configurator is a non-visual class that provides binding between the Business Control components and the rest of the Process Builder environment. It is used for managing Business Control properties, providing information about Business Control Editor (if it is enabled), and validating the configuration of the Business Control.

This class has to implement IBuisnessControlConfigurator. The following properties and methods should be implemented:

- Logic:
  - `ICollectionView<IODescriptor> NewInstanceInputs { get; }` – gets the Input descriptors required for a new instance of the Business Control
  - `ICollectionView<IODescriptor> NewInstanceOutputs { get; }` – gets the Output descriptors required for a new instance of the Business Control
  - `ICollectionView<IODescriptor> GetInstanceInputs(IBusinessControlProperties properties);` – gets the Inputs collection based on properties

- This method is used during the validation of the Business Control. The system checks if all of the Inputs exist and have the correct type. In a simple Business Control example, this could just return NewInstanceInputs, but in a more complicated situation it could be different.
- `ICollectionView<IODescriptor> GetInstanceOutputs(IBusinessControlProperties properties);` – gets the Outputs collection based on properties
- This method is used during the validation of the Business Control. The system checks if all of the Outputs exist and have the correct type. In a simple Business Control example, this could just return NewInstanceOutputs, but in a more complicated situation it could be different.
- `string EditorAssemblyName { get; }` – used to define the full assembly name
- `string EditorClassName { get; }` – used to define the name of the editor class
- `IBusinessControlProperties CreateProperties(FunctionDTO function, object serializedProperties);` – creates an IBusinessControlProperties object based on a serialized internal representation of the Business Control properties (refer to 4.1.2 Developing the Business Control Win Properties Class)
- `object SerializeProperties(IBusinessControlProperties properties);` – serializes the PB properties object to an internal representation of the Business Control properties
- `void OnInputAdded(FunctionInputDTO input);` – the method called after adding an Input to the Function
- `void OnInputRemoved(FunctionDTO function, FunctionInputDTO input);` – the method called after removing an Input from the Function
- `void OnInputChanged(FunctionInputDTO functionInput, ChangeType changeType, EventArgs args);` – the method called after changing an Input in the Function
- `void OnOutputAdded(FunctionOutputDTO output);` – the method called after adding an Output to the Function
- `void OnOutputRemoved(FunctionDTO function, FunctionOutputDTO output);` – the method called after removing an Output from the Function
- `void OnOutputChanged(FunctionOutputDTO functionOutput, ChangeType changeType, EventArgs args);` – the method called after changing an Output in the Function
- `void Validate(FunctionDTO function, BusinessControlFunctionPropertiesDTO controlGeneralProperties, Common.IServiceProvider serviceProvider, out IDictionary<IValidatableEntity, ICollection<Message>> validationMessagess);` – validates the Business Control properties
- ServiceProvider can be used in a situation when you want to get additional data from the server or call some methods on the server via remoting. In such cases, the interface should be added to the common shared `FlexNet.BusinessControlsSample.dll`. An additional project for service implementation can be added to the solution. Use of this service is shown below:

```
IFunctionsRepositoryService service =
serviceProvider.GetService<IFunctionsRepositoryService>();
                BusinessComponentMethodDTO bcmDTO =
service.GetBusinessComponentMethod(properties.BusinessComponentMethodId);
```

- Visual experience:
  - `System.Collections.Generic.ICollection<UnitType> SupportedSizeUnits { get; }` – used to define the units that can be used to describe the Business Control size
  - `CssStyle DefaultDesktopStyle { get; }` – used to define the default control rendering style on desktop devices
  - `CssStyle DefaultMobileStyle { get; }` – used to define the default control rendering style on mobile devices
  - `Unit? DefaultDesktopWidth { get; }` – used to define the default control width on desktop devices
  - `Unit? DefaultDesktopHeight { get; }` – used to define the default control height on desktop devices
  - `Unit? DefaultMobileWidth { get; }` – used to define the default control width on mobile devices
  - `Unit? DefaultMobileHeight { get; }` – used to define the default control height on mobile devices
  - `SizeF MinimumDesktopSize { get; }` – used to define the minimum control size on desktop devices
  - `SizeF MinimumMobileSize { get; }` – used to define the minimum control size on mobile devices
  - `System.Drawing.Image PreviewImage { get; }` – returns an image of the Business Control initially designed to be used in Layout Editor
  - `System.Drawing.Image Icon { get; }` – returns an image of the Business Control icon used in the toolbox in Layout Editor
  - `BusinessControlCategory Category { get; }` – defines the category of the Business Control and determines if and in which toolbox section the control will be displayed in Layout Editor
  - `BusinessControlType Type { get; }` – defines the type of the Business Control, and determines how the Business Control will be visualized in Layout Editor
  - `bool EditorNotRequired { get; }` – if true, then the Business Control properties editor is not needed and does not have to be specified

Example Code

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Web.UI.WebControls;
using System.Windows.Forms;
using FlexNet.DataServices.Process.Data;
using FlexNet.FunctionInterpreter.SystemServices.Transport;
using FlexNet.ProcessBuilder2.BusinessRules;
using FlexNet.ProcessBuilder2.BusinessRules.Function;
using FlexNet.ProcessBuilder2.BusinessRules.Function.FunctionProperties;
using FlexNet.DesignStudio.SystemServices.Validation;
using FlexNet.SystemServices;
using FlexNet.SystemServices.Collections;
using FlexNet.SystemServices.Utility;
using Image = System.Drawing.Image;
using Message = FlexNet.DesignStudio.SystemServices.Message;
using FlexNet.BusinessRules.BusinessControls2;

namespace FlexNet.BusinessControlsSample
{
    public class TestBusinessControlConfigurator : IBusinessControlConfigurator
    {
       //Optionaly instead of implementing IBusinessControlConfigurator you can derive
       //some methods already implemented.
       //ComponentBusinessControlConfigurator
       //UserInterfaceBusinessControlConfigurator
       public IBusinessControlProperties CreateProperties(FunctionDTO function, object
           serializedProperties)
       {
           BusinessControlFunctionPropertiesDTO bcpdto = function.Properties as
              BusinessControlFunctionPropertiesDTO;
           Assertion.ArgumentIsNotNull(bcpdto, "bcpdto");
           if (serializedProperties == null)
              return new TestBusinessControlPropertiesDTO();
           SampleProperties sampleProprties = Serializer.DeserializeFromXml
              ((string) serializedProperties, typeof(SampleProperties)) as
SampleProperties;
           return new TestBusinessControlPropertiesDTO(sampleProprties);
       }
       public object SerializeProperties(IBusinessControlProperties properties)
       {
            return Serializer.SerializeToXml((properties as
                TestBusinessControlPropertiesDTO).SampleProperties);
       }
       public IBusinessControlProperties DeserializeProperties(object serializedProperties)
       {
           return Serializer.DeserializeFromXml<TestBusinessControlPropertiesDTO>
(serializedProperties as string);
       }
       public System.Collections.Generic.ICollection<UnitType> SupportedSizeUnits
       {
           get { return null; }
```

```
        }
        public CssStyle DefaultDesktopStyle
        {
            get { return null; }
        }
        public CssStyle DefaultMobileStyle
        {
            get { return null; }
        }
        public Unit? DefaultDesktopWidth
        {
           get { return null; }
        }
        public Unit? DefaultDesktopHeight
        {
           get { return null; }
        }
        public Unit? DefaultMobileWidth
        {
           get { return null; }
        }
        public Unit? DefaultMobileHeight
        {
           get { return null; }
        }
        public SizeF MinimumDesktopSize
        {
           get { return SizeF.Empty; }
        }
        public SizeF MinimumMobileSize
        {
           get { return SizeF.Empty; }
        }

        //Getter should returns image. In our case it is added to project properties.
        //Add files to resources and change its name in code.
        public Image PreviewImage
        {
           get { return Resources.bcontrol_image_chart; }
        }

        //Getter should returns image. In our case it is added to project properties.
        //Add files to resources and change its name in code.
        public Image Icon
        {
           get { return Resources.bcontrol_toolbox_chart; }
        }
        public BusinessControlCategory Category
        {
           get { return BusinessControlCategory.Functional; }
        }
        public BusinessControlType Type
        {
           get { return BusinessControlType.UserInterface; }
        }
           public bool EditorNotRequired
        {
           get { return false; }
        }
        public string EditorAssemblyName
        {
           get
           {
               return
                   "FlexNet.BusinessControlsSampleWinUI, Version=1.0.0.0,Culture = neutral,
PublicKeyToken = 33f692327842122b";
           }
        }

        public string EditorClassName
        {
           get
           {
               return
                   "FlexNet.BusinessControlsSampleWinUI.TestBusinessControl.TestBusinessCont
rolEditor";
           }
        }

        public string HelpKeyword { get; private set; }

        public ICollectionView<IODescriptor> NewInstanceInputs
        {
           get
           {
               IList<IODescriptor> inputs = new List<IODescriptor>(
                   new[]
                   {
                       new IODescriptor("TestInput", "Test input description",
                           InputOutputType.Char)
                   }
               );
               return inputs;
           }
        }
        public ICollectionView<IODescriptor> NewInstanceOutputs
        {
           get
           {
               IList<IODescriptor> outputs = new List<IODescriptor>(
                   new[]
                  {
                      new IODescriptor("TestOutput", "Test output description",
                          InputOutputType.Char)
                  }
              );
              return outputs;
           }
        }

        public ICollectionView<IODescriptor> GetInstanceInputs(IBusinessControlProperties
properties)
        {
            TestBusinessControlPropertiesDTO testProperties =
                (TestBusinessControlPropertiesDTO) properties;
            if (testProperties.SampleProperties.TestInputRequired)
               return this.NewInstanceInputs;
            return null;
        }

        public ICollectionView<IODescriptor> GetInstanceOutputs(IBusinessControlProperties
            properties)
        {
            return this.NewInstanceOutputs;
        }

        public void Validate(FunctionDTO function, BusinessControlFunctionPropertiesDTO
            controlGeneralProperties, SystemServices.IServiceProvider serviceProvider, out
            IDictionary<IValidatableEntity, ICollection<Message>> validationMessagess)
        {
            validationMessagess = new Dictionary<IValidatableEntity, ICollection<Message>>
                ();
            IList<Message> collection = new List<Message>();
            validationMessagess.Add(function, collection);
            if (controlGeneralProperties.BusinessControlProperties == null)
            {
                collection.Add(Message.Error
                (ValidationMessages.BusinessControlPropertiesIsEmpty,
                    controlGeneralProperties.BusinessControl.Name));
                return;
            }

            try
            {
                SampleProperties properties =
                    ((TestBusinessControlPropertiesDTO)
controlGeneralProperties.BusinessControlProperties)
                    .SampleProperties;
                if (properties == null)
                {
                   collection.Add(Message.Error
                        (ValidationMessages.TestBusinessControlNotConfigured));
                }
            }
            catch
            {
                collection.Add(Message.Error
                     (ValidationMessages.TestBusinessControlNotConfigured));
            }
        }

        public void OnInputAdded(FunctionInputDTO input)
        {
        }

        public void OnInputRemoved(FunctionDTO function, FunctionInputDTO input)
        {
        }

         public void OnInputChanged(FunctionInputDTO functionInput, ChangeType changeType,
EventArgs args)
        {
        }

        public void OnOutputAdded(FunctionOutputDTO output)
        {
        }

        public void OnOutputRemoved(FunctionDTO function, FunctionOutputDTO output)
        {
        }

        public void OnOutputChanged(FunctionOutputDTO functionOutput, ChangeType changeType,
EventArgs args)
        {
        }
    }
}
```

4.1.2 Developing the Business Control Win Properties Class

The Business Control Properties class is a class with fields and get/set properties. Properties are used to persist into the database the information about the configuration options chosen on the Business Control configuration screen.

The class should implement the `IBusinessControlProperties` (which also implement `INotifyPropertyChanged` and `IUpdatable<IBusinessControlProperties>`) interface:

- `IBusinessControlProperties DettachedClone();`– this method returns a copy of the current properties and should create a standalone object of the Business Control properties without links to the Function
- `PropertyChangedEventHandler PropertyChanged;`– the event that is fired when a property of the Business Control changes
- `void UpdateFrom(IBusinessControlProperties source);`– this method updates the current properties according to the source properties

Example Code – SampleProperties Class

```
using System;
using System.ComponentModel;

namespace FlexNet.BusinessControlsSample
{
    [Serializable]
    public class SampleProperties : INotifyPropertyChanged
    {
        private string _test;
        public string TestProperty
        {
            get { return this._test; }
            set
            {
                if (value != this._test)
                {
                    this._test = value;
                    this.OnPropertyChanged("TestProperty");
                }
            }
        }

        private bool _testInputRequired = true;
        public bool TestInputRequired
        {
            get { return this._testInputRequired; }
            set
            {
                if (value != this._testInputRequired)
                {
                    this._testInputRequired = value;
                    this.OnPropertyChanged("TestInputRequired");
                }
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;
        protected void OnPropertyChanged(string propertyName)
        {
            if (this.PropertyChanged != null)
                this.PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Example Code – TestBusinessControlPropertiesDTO Class

```
using System.ComponentModel;
using FlexNet.Common;
using FlexNet.ProcessBuilder2.BusinessRules.Function;

namespace FlexNet.BusinessControlsSample
{
    public class TestBusinessControlPropertiesDTO : IBusinessControlProperties
    {
        public SampleProperties SampleProperties { get; set; }

        public TestBusinessControlPropertiesDTO(SampleProperties sampleProperties)
        {
            this.SampleProperties = sampleProperties;
        }

        public TestBusinessControlPropertiesDTO()
        {
            this.SampleProperties = new SampleProperties();
        }

        [DTOIgnore]
        public event PropertyChangedEventHandler PropertyChanged
        {
            add { this.SampleProperties.PropertyChanged += value; }
            remove { this.SampleProperties.PropertyChanged -= value; }
        }

        public void UpdateFrom(IBusinessControlProperties source)
        {
            if (source as TestBusinessControlPropertiesDTO == null)
                return;

            TestBusinessControlPropertiesDTO sourceProperties =
(TestBusinessControlPropertiesDTO)source;

            this.SampleProperties.TestProperty =
sourceProperties.SampleProperties.TestProperty;
            this.SampleProperties.TestInputRequired =
sourceProperties.SampleProperties.TestInputRequired;
        }

        public IBusinessControlProperties DettachedClone()
        {
            TestBusinessControlPropertiesDTO properties = new
TestBusinessControlPropertiesDTO(new SampleProperties());
            properties.UpdateFrom(this);
            return properties;
        }
    }
}
```

Example Code – Validation of Messages Literals

```
using FlexNet.SystemServices;
namespace FlexNet.BusinessControlsSample
{
    [LiteralDefinition]
    public enum ValidationMessages
    {
        [LiteralDefinition("Business Control Properties is empty.")]
        BusinessControlPropertiesIsEmpty,
        [LiteralDefinition("Business Control is not configured.")]
        TestBusinessControlNotConfigured
    }
}
```

4.2 Developing the Business Control Runtime Project

To create the project, perform the following tasks:

1. Create an ASP.NET Web Application project. The example project is named **BusinessControlsSampleWebUI**.
2. Add a folder for the Business Control that is created.
3. Add the following classes to the folder:
   a. <control name> BusinessControl class. For instructions on creating this class, refer to 4.2.1 Developing the Business Control Main Class.
   b. <control name> Runtime Web user control class. For instructions on creating this class, refer to 4.2.2 Developing the Business Control Runtime Control.
   c. <control name> RuntimeMobile mobile Web user control class (if the control is supposed to work on mobile devices). For instructions on creating this class, refer to 4.2.2 Developing the Business Control Runtime Control.
   d. <control name> RuntimeText mobile Web user control class (if the control is supposed to work on text devices and the mobile control is different than text). For instructions on creating this class, refer to 4.2.2 Developing the Business Control Runtime Control.
4. Add references for
   - `FlexNet.FunctionInterpreter.BusinessRules.Functions.dll`
   - `FlexNet.FunctionInterpreter.SystemService.Transport.dll`
   - `FlexNet.SystemServices.dll`
   - `FlexNet.DataServices.Process.Data.dll`
   - `FlexNet.WebUI.dll`
   - `FlexNet.WebUI.BusinessControls.dll`
   - `BusinessControlsSample project`

   The project tree should look similar to the one shown in the figure below.

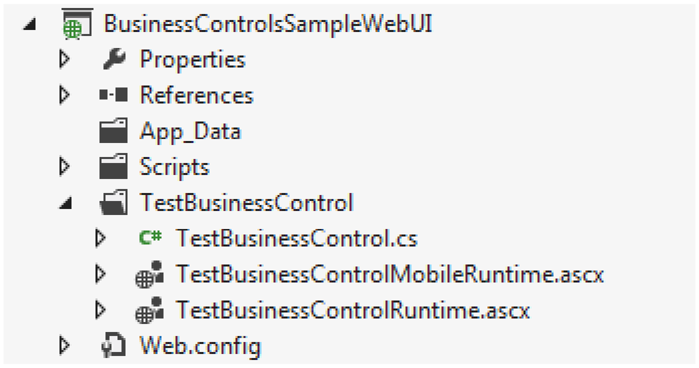

BusinessControlsSampleWebUI
Properties
References
App_Data
Scripts
TestBusinessControl
TestBusinessControl.cs
TestBusinessControlMobileRuntime.ascx
TestBusinessControlRuntime.ascx
Web.config

Figure 2 Sample Business Control WebUI Project Tree

4.2.1 Developing the Business Control Main Class

The Main Business Control class is a non-visual class that provides binding between the Business Control components and the rest of the DELMIA Apriso environment. It contains the information necessary for the Business Control to be registered in the Business Component Repository as well as the names of the configuration and runtime controls of which the Business Control consists. It is also used to validate the configuration data entered by the user in DELMIA Apriso Process Builder, generate PB Function Inputs and Outputs, or execute custom logic in runtime.

The Main Business Control class has to be marked with two attributes:

- *ComponentRepositoryComponentType*

  This is the attribute used by the Component Repository engine to register the control. See an example below.

  ```
  [ComponentRepositoryComponentType(ComponentType.BusinessControl, “NEW_FUID” )]
  ```

  The example attribute contains two required parameters:
  - ComponentType
  - FUID – a unique identifier used in the DELMIA Apriso system that can be generated using tools like a GUID generator

- *BusinessControl*

  This is the attribute that supplies information about the Business Control. See an example below.

```
[BusinessControl("SampleBusinessControl",
    "1.0",
    "SampleBusinessControlDescription",
    "SampleBusinessControlRuntime.ascx",
    "SampleBusinessControlMobileRuntime.ascx",
    null,
    "FlexNet.BusinessControlsSampleWinUI, Version=9.4.0.0, Culture=neutral,
PublicKeyToken=33f692327842122b",
"FlexNet.BusinessControlsSampleWinUI.TestBusinessControl.TestBusinessControlConfigurato
r")]
```

The example attribute contains the following parameters:

- `SampleBusinessControl` – the name of the Business Control
- `1.0` – the version of the Business Control
- `SampleBusinessControlDescription` – the description of the Business Control
- `SampleBusinessControlRuntime.ascx`, `SampleBusinessControlMobileRuntime.ascx`, `null` – the paths to the desktop, mobile and text runtime controls (if a control is not used, insert null)
- `FlexNet.BusinessControlsSampleWinUI, Version=9.4.0.0, Culture=neutral, PublicKeyToken=33f692327842122b` – the details of the DLL that contains the Win Configurator
- `FlexNet.BusinessControlsSampleWinUI.TestBusinessControl.TestBusinessControlConfigurator` – the name of the Win Configurator class

It is very important to specify the file locations as relative paths to the main Business Control path. For example:

```
[BusinessControl
("Checklist Control", "1.0",
"Displays a checklist and persists a user's answers to the database",
"Checklist/CheckListRuntime.ascx", null, null,
"FlexNet.ProcessBuilder2.BusinessRules, Version=9.4.0.0, Culture=neutral,
PublicKeyToken=33f692327842122b",
"FlexNet.ProcessBuilder2.BusinessRules.Function.Configurators.BusinessControls.CheckList
Configurator")]
```

The Main Business Control class also needs to inherit from `FlexNet.WebUI.BusinessControls. BusinessControl` (override methods if needed) or implement the IBusinessControl interface:

- Outcome ExecutePreRenderLogic(object properties, PropertyBag inputs, PropertyBag sessionVariables) – executed right before the runtime engine renders the Business Control to the user (this method is executed inside the Function Interpreter database transaction!)
- Outcome ExecutePostRenderLogic(object properties, PropertyBag outputs, PropertyBag sessionVariables) – executed just after the Operation screen is submitted (this method is executed inside the Function Interpreter database transaction!)

Some interface methods are marked as obsolete and will be removed from the interface in the next version (these methods are not required, because Web Process Builder has been removed):

- InputOutputDescriptorCollection GetInputsDefinition(object properties)
- InputOutputDescriptorCollection GetOutputsDefinition(object properties)
- OutcomeCollection Validate(object properties, InputOutputDescriptorCollection inputs, InputOutputDescriptorCollection outputs, CssStyleSet style)

Example Code

```
using FlexNet.BusinessControlsSample;
using FlexNet.DataServices.Process.Data;
using FlexNet.FunctionInterpreter.BusinessRules.Functions;
using FlexNet.SystemServices;
using FlexNet.WebUI.BusinessControls;

namespace FlexNet.BusinessControlsSampleWebUI.TestBusinessControl
{
    [ComponentRepositoryComponentType(ComponentType.BusinessControl, "FA7A0A07-3341-4c41-
ACFE-7BDB59AAE9FB")]
    [BusinessControl("SampleBusinessControl",
    "1.0",
    "TestBusinessControlDescription",
    "TestBusinessControlRuntime.ascx",
    "TestBusinessControlMobileRuntime.ascx",
    null,
    "FlexNet.BusinessControlsSample, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=33f692327842122b",
    "FlexNet.BusinessControlsSample.TestBusinessControlConfigurator")]
    public class TestBusinessControl : BusinessControl
    {
        public override InputOutputDescriptorCollection GetInputsDefinition(
            object properties)
        {
            InputOutputDescriptorCollection result = new InputOutputDescriptorCollection
                {
                    new InputOutputDescriptor("TestInput", "TestInputDescription",
InputOutputType.Char)
                };

            return result;
        }

        protected override InputOutputDescriptorCollection GetRequiredInputs(object
properties)
        {
            SampleProperties props = (SampleProperties)properties;
            if (props.TestInputRequired)
                return this.GetInputsDefinition(properties);

            return new InputOutputDescriptorCollection();
        }

        public override InputOutputDescriptorCollection GetOutputsDefinition(object
properties)
        {
            InputOutputDescriptorCollection result = new InputOutputDescriptorCollection
                {
                    new InputOutputDescriptor("TestOutput", "TestOutputDescription",
InputOutputType.Char)
                };
            return result;
        }
    }
}
```

4.2.2 Developing the Business Control Runtime Control

Business Control Runtime is a set of ASP.NET user Web controls which are used by Function Interpreter during execution to display the Business Control.

The runtime control needs to implement the IBusinessControlRuntime interface:

- `void Initialize(object properties, PropertyBag inputs, PropertyBag sessionVariables, CssStyleSet style)` – the initialized Business Control just before it is shown to the user, and this is invoked only once
- `Outcome Validate()` – validates if the data on the screen is correct (e.g., if the Inputs have the correct types, etc.)
- `void GetOutputsValues(PropertyBag outputs)` – gets a Function Outputs based on values from the screen, and this is invoked once when the page is submitted (for example, the user clicks the OK button or invokes SubmitPage event [see below])
- `event EventHandler SubmitPage` – could be used to submit the page (without clicking the OK button)

The desktop control must inherit from **FlexNet.WebUI.UserControl**.

> The runtime control must be composed of ASP.NET controls. You should avoid using static HTML controls that are the same as static string value assignments in runtime.

Example Code

```
using System;
using FlexNet.BusinessControlsSample;
using FlexNet.FunctionInterpreter.BusinessRules.Functions;
using FlexNet.FunctionInterpreter.SystemServices.Transport;
using FlexNet.SystemServices;
using FlexNet.WebUI;

namespace FlexNet.BusinessControlsSampleWebUI.TestBusinessControl
{
    public partial class TestBusinessControlRuntime : UserControl, IBusinessControlRuntime
    {
        protected System.Web.UI.WebControls.TextBox TextBox1;
        protected System.Web.UI.WebControls.TextBox TextBox2;
        protected System.Web.UI.WebControls.Label Label1;
        protected System.Web.UI.WebControls.Label Label2;
        //-----------------------------------------------------------------------------
        // Additional code
        //-----------------------------------------------------------------------------

        public void GetOutputsValues(PropertyBag outputs)
        {
            outputs["TestOutput"] = this.TextBox1.Text + ":" + this.TextBox2.Text;
        }

        public void Initialize(object properties, PropertyBag inputs, PropertyBag
sessionVariables, CssStyleSet style)
        {
            SampleProperties sampleProperties = Serializer.DeserializeFromXml
(properties.ToString(), typeof(SampleProperties)) as SampleProperties;

            if (sampleProperties.TestInputRequired)
                this.TextBox1.Text = sampleProperties.TestProperty + ":" + inputs
["TestInput"];
            else
                this.TextBox1.Text = sampleProperties.TestProperty;
        }

        public Outcome Validate()
        {
            return Outcome.Success();
        }

        public event EventHandler SubmitPage;
    }
}
```

Runtime ASCX file

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="TestBusinessControlRuntime.ascx.cs"
Inherits="FlexNet.BusinessControlsSampleWebUI.TestBusinessControl.TestBusinessControlRuntim
e, FlexNet.BusinessControlsSampleWebUI, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=33f692327842122b"
TargetSchema="http://schemas.microsoft.com/intellisense/ie5"%>
<table>
<tr>
<td>
            <asp:Label id="Label1" runat="server">Properties : Test Input</asp:Label>
        </td>
    <td>
            <asp:TextBox id="TextBox1" runat="server" Enabled="False"></asp:TextBox>
        </td>
    </tr>
<tr>
<td>
            <asp:Label id="Label2" runat="server">Test Output</asp:Label>
        </td>
<td>
            <asp:TextBox id="TextBox2" runat="server"></asp:TextBox>
        </td>
</tr>
</table>
```

4.3 Developing the Business Control Win Editor Project

If your control does not require a visual editor, you do not have to create this project.

To create the project, perform the following tasks:

1. Create a Class Library project. The example project is named **BusinessControlsSampleWinUI**.
2. Add a folder for the Business Control that will be created.
3. Add the following class to the folder: <control name> Business Control Editor class (if required). This should be User Control class.
4. Add reference for:
   - `FlexNet.DataServices.Process.Data.dll`
   - `FlexNet.DesignStudio.WinUI.dll`
   - `FlexNet.DesignStudio.SystemServices.dll`
   - `FlexNet.ProcessBuilder2.BusinessRules.dll`
   - This assembly contains all of the required interfaces for the Win Configurator control
   - `FlexNet.ProcessBuilder2.WinUI.dll`
   - `FlexNet.SystemServices.dll`
   - `System.Drawing.dll`
   - `System.Web.dll`
   - `BusinessControlsSample project`

The project tree should look similar to the one shown in the figure below.

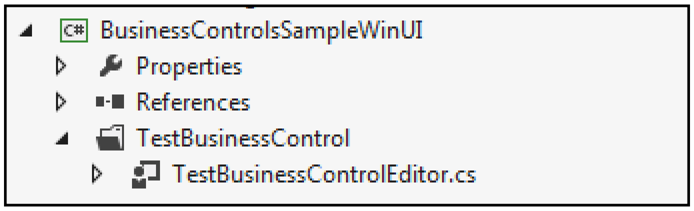


Figure 3 Sample Business Control WinUI Project Tree

4.3.1 Developing the Business Control Win Editor Class

If it is required to implement the Editor class. The Business Control Editor for DELMIA Apriso Process Builder should be derived from the UserControl class and has to implement the `IBusinessControlEditor` interface.

- `void SetReadOnly(bool readOnly);` – implements the read-only property on all controls when they are called
- `void EditFunction(FunctionDTO function);` – sets all the properties on the editor based on information from the Function
- `void CompleteEditing();` – releases all of the objects that are not needed anymore

Example Code

```
using System;
using System.Windows.Forms;
using FlexNet.BusinessControlsSample;
using FlexNet.DataServices.Process.Data;
using FlexNet.DesignStudio.WinUI;
using FlexNet.ProcessBuilder2.BusinessRules.Function;
using FlexNet.ProcessBuilder2.BusinessRules.Function.FunctionProperties;
using FlexNet.ProcessBuilder2.WinUI.Functions.Actions;
using FlexNet.ProcessManager.Model.Function;
using FlexNet.ProcessManager.Security;
using SecurityManager = FlexNet.ProcessManager.Security.SecurityManager;

namespace FlexNet.BusinessControlsSampleWinUI.TestBusinessControl
{
    public partial class TestBusinessControlEditor : UserControl, IBusinessControlEditor
    {
        private bool _settingControls;
        private FunctionDTO _editedFunction;

        private SampleProperties _sampleProperties;

        public TestBusinessControlEditor()
        {
            InitializeComponent();
        }


        public void EditFunction(FunctionDTO function)
        {
            this._editedFunction = function;
            this._sampleProperties = ((TestBusinessControlPropertiesDTO)
((BusinessControlFunctionPropertiesDTO)this._editedFunction.Properties).
                        BusinessControlProperties).SampleProperties;
            this.SetValuesToControls(this._sampleProperties);
            this.SetReadOnly(this._editedFunction.IsReadOnly);
            this._editedFunction.ReadOnlyChanged += this.EditedFunctionReadOnlyChanged;
            this._sampleProperties.PropertyChanged += this.SamplePropertiesPropertyChanged;
        }

        public void SetReadOnly(bool readOnly)
        {
            AccessLevel uiAccessLevel = SecurityManager.HasAccess
(Permission.UIControlProperties);
            if (uiAccessLevel == AccessLevel.None || uiAccessLevel == AccessLevel.ReadOnly)
                readOnly = true;

            this.textBox1.Enabled = !readOnly;
            this.checkBoxTestInputRequired.Enabled = !readOnly;
        }

        public void CompleteEditing()
        {
            this._editedFunction.ReadOnlyChanged -= this.EditedFunctionReadOnlyChanged;
            this._sampleProperties.PropertyChanged -= this.SamplePropertiesPropertyChanged;
            this._sampleProperties = null;
        }


        private void SetValuesToControls(SampleProperties properties)
        {
            this._settingControls = true;

            this.textBox1.Text = properties.TestProperty ?? string.Empty;
            this.checkBoxTestInputRequired.Checked = properties.TestInputRequired;

            this._settingControls = false;
        }

        void SamplePropertiesPropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
        {
            this.SetValuesToControls(this._sampleProperties);
        }

        void EditedFunctionReadOnlyChanged(object sender,
SystemServices.ValueChangedEventArgs<bool> e)
        {
            this.SetReadOnly(this._editedFunction.IsReadOnly);
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            if (this._settingControls)
                return;

            this._sampleProperties.TestProperty = this.textBox1.Text;
        }

        private void checkBoxTestInputRequired_CheckedChanged(object sender, EventArgs e)
        {
            if (this._settingControls)
                return;

            this._sampleProperties.TestInputRequired =
this.checkBoxTestInputRequired.Checked;

            if (this._sampleProperties.TestInputRequired && this._
editedFunction.ContainsInputName("TestInput") == false)
            {
                Workbench.Current.ActiveDocument.Execute(new AddInputAction(this._
editedFunction, FunctionInputSourceType.Constant, InputOutputType.Char, "TestInput", "Test
input description", false));
            }
        }
```

```
    }
}
```

4.4 Using a Business Control in DELMIA Apriso

To understand the role of each element of a Business Control, it is necessary to know how a Business Control is installed on a running Apriso server and how users interact with it.

4.4.1 Installing the Business Control on a DELMIA Apriso Server

As a result of the development of a Business Control, several ASCX files (ASP.NET user controls, e.g., `TestBusinessControlRuntime.ascx`) and DLL files (e.g., `FlexNet.BusinessControlsSampleWebUI.dll`, `FlexNet.BusinessControlsSampleWinUI.dll`, and `FlexNet.BusinessControlsSample.dll`) are created. They contain the logic of the Business Control (for instructions on creating these assemblies, refer to 4.1 Developing the Business Control Properties Project, 4.2 Developing the Business Control Runtime Project and 4.3 Developing the Business Control Win Editor Project):

- The DLLs need to be added to .NET Global Assembly Cache of the server or folder from where ClickOnce application is downloaded:
  - `FlexNet.BusinessControlsSampleWebUI.dll` should be added to the .NET Global Assembly Cache
  - `FlexNet.BusinessControlsSampleWinUI.dll` should be added to the installation folder of the Click Once application (by default `<drive>\Program Files\Dassault Systemes\DELMIA Apriso 2021\WebSite\Downloads\PB2`)
  - `FlexNet.BusinessControlsSample.dll` should be added to both the .NET Global Assembly Cache and the ClickOnce installation folder
- The ASCX files should be copied to a new subfolder of the BusinessControls folder of the Portal applications (by default `<drive>\Program Files\Dassault Systemes\DELMIA Apriso 2021\WebSite\Portal\BusinessControls\<control subfolder name>`)
- This can be changed in Central Configuration using `BusinessControlsPath` key located in the “FunctionInterpreter” section (for details, see Central Configuration Documentation)
- The ClickOnce application manifest should be regenerated. To do this, the `Publish Apriso Process Builder via ClickOnce.bat` batch file should be executed (by default this is located in the `<drive>\Program Files\Dassault Systemes\DELMIA Apriso 2021\WebSite\Downloads\ClickOnce Tools` folder)
- The next step requires registering the Business Control in the Component Repository under the “Business Control” type
  - After successful registration, the control is ready to use
  - If you cannot see your control on the Business Control list when creating a new Function, it may sometimes be required to restart the ProcessBuilder service and re-open the PB application

4.4.2 Configuring the Business Control

All Business Controls are configured in DELMIA Apriso Process Builder. There is a special type of Function designed for configuring Business Controls. To include a control in an Operation, add a Function of the Business Control type from the Toolbox to one of its Steps.

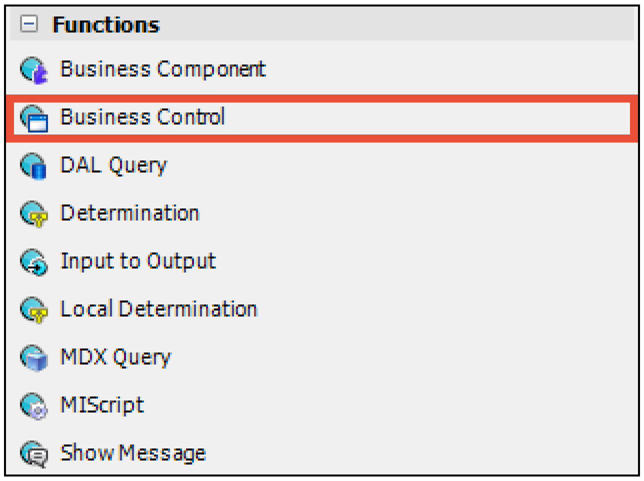


Figure 4 Business Control function in Process Builder

For instructions on managing Steps and Functions in Process Builder, refer to Process Builder Help.

4.4.3 The Business Control in Runtime

After releasing the Operation which includes the Business Control, Function Interpreter will render the control using the configured properties and the current Inputs. For the user of the Operation, using the Business Control does not differ in any way from using other Functions configured in Process Builder.

5 Best Practices

5.1 Web Controls (*.ascx)

Use the full assembly name in ASCX files

Each `*.ascx` file contains a reference to its DLL, for example:

```
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="CheckListConfig.ascx.cs"
Inherits="FlexNet.WebUI.BusinessControls.Checklist.CheckListConfig,
FlexNet.WebUI.BusinessControls, Version=9.0.0.0, Culture=neutral,
PublicKeyToken=33f692327842122b"
TargetSchema="http://schemas.microsoft.com/intellisense/ie5"%>
```

The inherits tag should appear as “class name (with namespace), full assembly name” – this is required if the DLL is in GAC and not in the Web application bin folder!

5.2 Shared Web Controls

Shared controls overview

During the development of the first set of Business Controls, some parts of runtime screens were repeating more than once. That led to the creation of several ASP.NET controls that may be used as parts of the new Business Control screens. Currently these are: InlineSidebar.

Using these controls is strongly encouraged whenever possible, as it prevents duplication of the code and ensures the unified look and feel of all Business Control screens.

The controls need to be referenced in the `*.ascx` file, for example:

```
<%@ Register TagPrefix="uc1" TagName="InputOutputMapper" Src="~/InputOutputMapper.ascx" %>
```

The existing shared controls should be used in the following situations:

- **InlineSidebar**

  This control can be used only by runtime controls. Given the ID of the HTML control (this control has a child of an HTML element that can contain a <table> element object), it wraps that control’s content and displays a sidebar next to it. Check the usage of the InlineSidebar control in the Work Instructions Business Control.

5.3 Shared Actions

When developing the Process Builder Configurator or Editor, you have to use actions for all the Operations that change the status of an Input, Output, or Routing. Actions must be classes that implement the IAction interface, so that they support the undo/redo functionality. This is required because in the case of calling an undo operation for another action, the state of an object could be inconsistent and produce an error.

In the `FlexNet.ProcessBuilder2.WinUI.Functions.Actions` namespace, there are predefined actions which allow for manipulating Function properties. See the example list below:

- AddFunctionAction
- AddInputAction
- AddOutputAction
- AddOutputAndLinkToInputAction
- AddOutputRoutingAction
- ChangeFunctionTypeAction
- ChangeInputDataTypeAction
- CreateSVRoutingToExistingOutputAction
- RemoveInputAction
- RemoveOutputAction
- RemoveOutputRoutingAction
- ResequenceInputAction

You can either use the predefined actions or create your own actions. Using existing actions is very simple, as in this example:

```
private void checkBoxEmployeeNoList_CheckedChanged(object sender, System.EventArgs e)
{
   if (this._settingControls)
      return;

   AddRemoveEmployeeNoInputAction action = new AddRemoveEmployeeNoInputAction(this._
editedFunction, this.checkBoxEmployeeNoList.Checked);
   Workbench.Current.ActiveDocument.Execute(action);
}
```

The code is calling an action that creates or removes the *EmployeeNo* input in a Function based on the selected property (check box).

It is also possible to create a Composite Action that will call many other simple actions:

```
List<IAction> actions = new List<IAction>();
foreach (InputOutputDescriptor descriptor in this._requiredInputs)
{
if (this._editedFunction.Inputs[descriptor.Name] == null)
actions.Add(new AddInputAction(this._editedFunction, FunctionInputSourceType.Constant,
descriptor.Type, descriptor.Name));
}
IAction addInputAction = new CompositeAction(actions);
   Workbench.Current.ActiveDocument.Execute(addInputAction);
```

Starting the action cannot be done by just calling `addInputAction.Do();`, because in such a case there will be no support for undo/redo. You have to use the `Workbench.Current.ActiveDocument.Execute(addInputAction);` command to start the execution.

5.4 Debugging Business Control Editor Class

If you have an editor that is a bit more complicated, you will sometimes want to debug it. The best way to do this is to copy PB locally to the disk, and then copy the new Business Control Editor DLL to this folder as well as the PDB file. Start PB from this folder and attach to this Process with your Business Control solution.

It is very important to make sure that you do not have debugging the DLL in GAC. In such a case, any changes applied in the solution will not be visible, because the DLL is read from GAC first. That is why it is not recommended to develop a Business Control solution on a computer which is also the DELMIA Apriso server.

6 References

Internal Documentation

1. ***Process Builder Help***

   Provides an overview of DELMIA Apriso Process Builder (PB) and information on installing and using the application. This Help describes the user interface elements, entity maintenance, available Business Controls, and management of Processes, Operations, and Screen Flows.

2. ***Central Configuration Documentation***

   Describes in detail all the keys of the Central Configuration (CC) file for DELMIA Apriso. Various sections group the keys for individual modules or distinct functional areas.

3DS Support Knowledge Base

If you have any additional questions or doubts not addressed in our documentation, feel free to visit the **3DS Support Knowledge Base** at https://support.3ds.com/knowledge-base/.