

Ribbon Designer

for SharePoint® and Office 365

DEVELOPER'S GUIDE

Add-in Express™

www.add-in-express.com

Ribbon Designer for SharePoint® and Office 365

Developer's Guide

Revised on 15-Mar-12

Copyright © Add-in Express Ltd. All rights reserved.

Add-in Express, ADX Extensions, ADX Toolbar Controls, Afalina, AfalinaSoft and Afalina Software are trademarks or registered trademarks of Add-in Express Ltd. in the United States and/or other countries. Microsoft, Outlook, and the Office logo are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries. Borland and the Delphi logo are trademarks or registered trademarks of Borland Corporation in the United States and/or other countries.

THIS SOFTWARE IS PROVIDED "AS IS" AND ADD-IN EXPRESS LTD. MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, ADD-IN EXPRESS LTD. MAKES NO REPRESENTATIONS OR WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE LICENSED SOFTWARE, DATABASE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS

Why Ribbon Designer?	5
Trial Version	5
System Requirements	5
Technical Support	5
Installing and Activating	6
Activation Basics.....	6
Setup Package Contents	7
Solving Installation Problems.....	7
Getting Started	8
Your First SharePoint Ribbon Tab	9
A Bit of Theory	9
Adding the Ribbon Designer	9
Ribbon Designer Basics.....	13
Ribbon Designer Files	14
Ribbon Tabs and Controls	14
Server-side Events	16
Client-side Events	18
Debugging the Project	20
Deploying the Ribbon UI	20
Deploying for Office 365	20
What's next?	20
Using the Ribbon Designer	21
Ribbon Controls in Detail	22
Creating and Showing Ribbon Contextual Groups	22
Ribbon Control Visibility Rules	23
How Your Control is Displayed	23
Ribbon Designer Programmability	24
Localization.....	24
Accessing Ribbon Controls in JavaScript	24
Creating Ribbon Controls Dynamically	24
Preserving the State of Ribbon Controls.....	26
Ribbon Designer Events	26
Dealing with Built-in Ribbon Controls	28
Add a custom tab.....	28
Position a custom tab	29
Add a custom group to a built-in tab	30
Add a custom control to a built-in group	31
Position a custom control in a built-in group	32
Customize a built-in button	33
Disable a built-in control.....	34
Hide a built-in control	35
Hide a built-in group.....	36
Hide a built-in tab.....	37
Use a built-in control on a custom control container	37
Disable a built-in group or tab.....	37
Integrating with Existing Solutions	38
The Ribbon UI and your SharePoint application	38
Importing Existing Ribbon Controls.....	40

Introduction

The Ribbon Designer for SharePoint® and Office 365 is a development tool designed to simplify and speed up the development of the Ribbon UI in Visual Studio® 2010 or Visual Studio 11 Beta through the consistent use of the RAD paradigm. It provides a number of specialized components allowing the developer to skip the interface-programming phase and get to functional programming in no time.

Why Ribbon Designer?

The Ribbon Designer simplifies customizing and extending the Ribbon UI of SharePoint and Office 365 using VB.NET and C# on Visual Studio 2010 or Visual Studio 11 Beta.

The Ribbon Designer allows you to overcome the basic problem of customizing the Server Ribbon when working with SharePoint objects or Office 365 applications – getting to the required UI quickly. Based on the True RAD paradigm, the Ribbon Designer saves the time and effort that you would have to spend on research, prototyping and debugging numerous issues when creating the Server Ribbon. The issues include creating a ribbon markup complying with the Ribbon rules – the process requires learning the features defined by the Ribbon schema. The Ribbon Designer also allows handling Ribbon events, both server and JavaScript, as well as connecting JavaScript code to the solution and debugging it.

Trial Version

If you use the trial version, you can create as many tabs, groups and controls at design-time as you need but only one group will be shown at run time.

System Requirements

Supported SharePoint editions

- Microsoft SharePoint 2010 Foundation
- Microsoft SharePoint 2010 Standard
- Microsoft SharePoint 2010 Enterprise
- Microsoft SharePoint Online

Supported Office 365 Subscription Plans

- Office 365 for professionals and small businesses (Plan P)
- Office 365 for midsize businesses and enterprises (Plans E1, E2, E3, E4)

Programming languages

- Visual C#
- Visual Basic
- JavaScript (for client-side Ribbon events)

Visual Studio versions

- Visual Studio 2010 with SharePoint Developer Tools installed
- Visual Studio 11 Beta

Technical Support

The Ribbon Designer for SharePoint and Office 365 is developed and supported by the Add-in Express Team, a branch of Add-in Express Ltd. You find the Add-in Express web site at www.add-in-express.com.

For technical support through the Internet use our [forums](#) or e-mail us at support@add-in-express.com.

Installing and Activating

There are two key points in the product installation. First off, you have to specify the development environments in which you are going to use the Ribbon Designer (see [System Requirements](#)). Secondly, you need to activate the product.

Activation Basics

During the registration process, the registration wizard prompts you to enter your license key. The key is a 30-character alphanumeric code shown in six groups of five characters each (for example, AXN4M-GBFTK-3UN78-MKF8G-T8GTY-NQS8R). Keep the license key in a safe location and do not share it with others. This product key forms the basis for your ability to use the software.

For purposes of product activation only, a non-unique hardware identifier is created from general information that is included in the system components. At no time are files on the hard drive scanned, nor is personally identifiable information of any kind used to create the hardware identifier. Product activation is completely anonymous. To ensure your privacy, the hardware identifier is created by what is known as a "one-way hash". To produce a one-way hash, information is processed through an algorithm to create a new alphanumeric string. It is impossible to calculate the original information from the resulting string.

Your product key and a hardware identifier are the only pieces of information required to activate the product. No other information is collected from your PC or sent to the activation server.

If you choose the *Automatic Activation Process* option of the activation wizard, the wizard attempts to establish an online connection to the activation server, www.activatenow.com. If the connection is established, the wizard sends both the license key and the hardware identifier over the Internet. The activation service generates an activation key using this information and sends it back to the activation wizard. The wizard saves the activation key to the registry.

If an online connection cannot be established (or you choose the *Manual Activation Process* option), you can activate the software using your web-browser. In this case, you will be prompted to enter the product key and a hardware identifier on a web page, and you will get an activation key. This process finishes with saving the activation key to the registry.

Activation is completely anonymous; no personally identifiable information is required. The activation key can be used to activate the product on that computer an unlimited number of times. However, if you need to install the product on several computers, you will need to perform the activation process again on every PC.

Please refer to your end-user license agreement for information about the number of computers you can install the software on.

Setup Package Contents

The Ribbon Designer for SharePoint and Office 365 setup program installs the following folders on your PC:

- *Bin* – binary files
- *Docs* – documentation including the class reference
- *Images* – icons
- *Sources* – source code (see the note below)

Please note that the source code of the Ribbon Designer is or is not delivered depending on the product package you purchased. See the section [Editions and prices](#) on our web site for details.

The Ribbon Designer for SharePoint and Office 365 setup program installs the following text files on your PC:

- *licence.txt* – EULA
- *readme.txt* – short description of the product, support addresses and such
- *whatsnew.txt* – this file contains the latest information on the new features and fixed bugs.

Solving Installation Problems

Make sure you are an administrator on the PC.

Remove the following registry key, if it exists:

```
HKEY_CURRENT_USER\Software\Add-in Express\Ribbon Designer for SharePoint and Office 365 {package}
```

On Vista, Windows 7 and Windows 2008 Server, set UAC to its default level.

In *Control Panel | System | Advanced | Performance | Settings | Data Execution Prevention*, set the "... for essential Windows programs and services only" flag.

Run *setup.exe*, not the *.MSI*.

Finally, use the *Automatic Activation Process* option in the installer windows.

Getting Started

Here we guide you through the main steps of using the Ribbon Designer in your projects.

Your First SharePoint Ribbon Tab

The sample project below demonstrates how you create the Ribbon UI of your SharePoint or Office 365 solution in Visual Studio 2010.

A Bit of Theory

In SharePoint Foundation terms, the Ribbon Designer is a *feature*. It provides a True RAD way for developing a custom Ribbon UI of your SharePoint solution. Using information provided at design-time, the Ribbon Designer creates a well-formed XML markup, which complies with the Server Ribbon scheme. Then, at a proper moment, the Ribbon Designer sends the XML to the server. Basing on the markup, the server creates the required Ribbon UI and connects it to the events - both server and client.

*Developing SharePoint projects requires that you have **administrative permissions**. In addition, if you have Windows Vista, Windows 7, or Windows 2008, it is imperative that you start Visual Studio 2010 (Visual Studio 11 Beta) via **Run as Administrator**.*

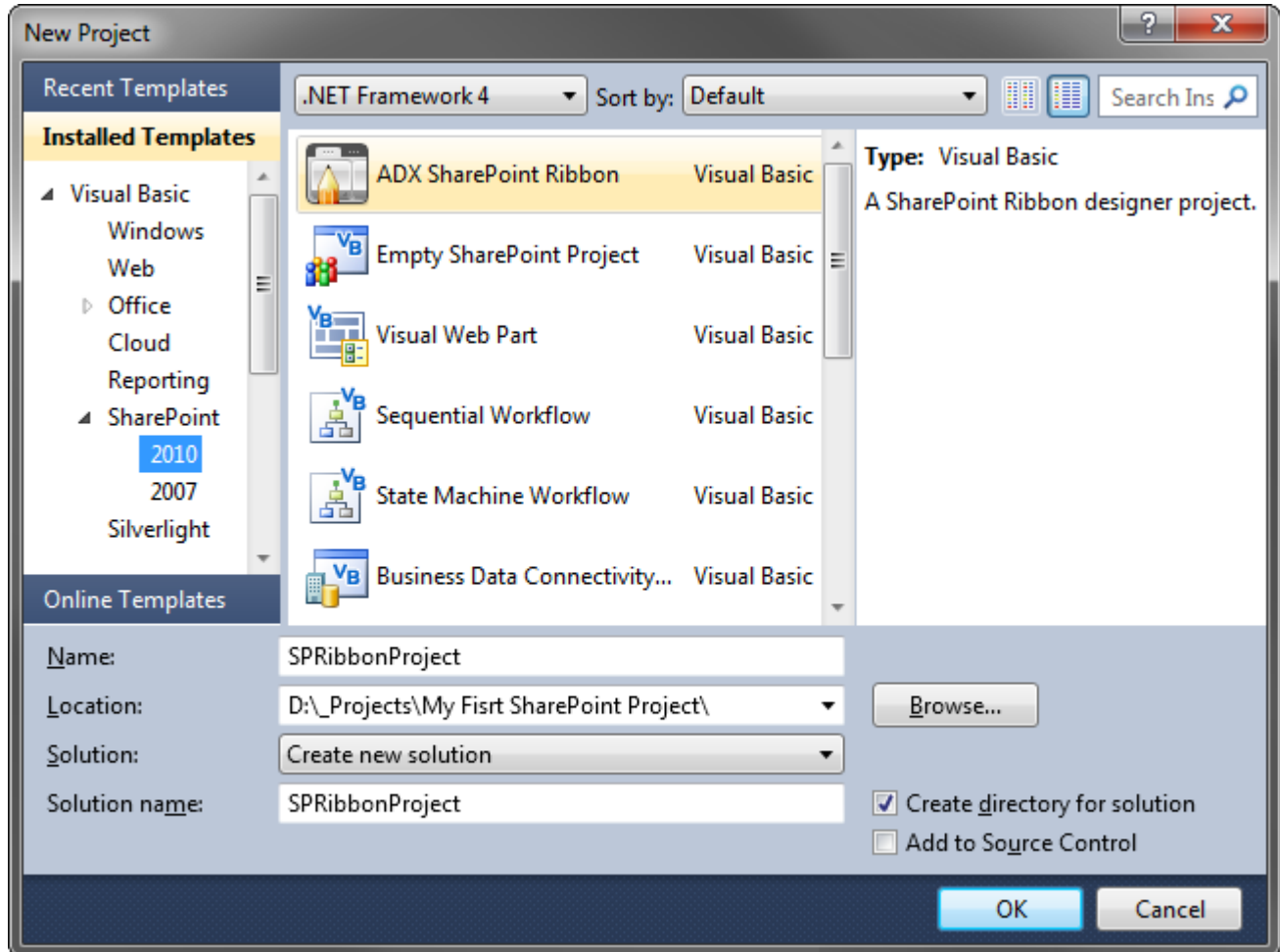
Adding the Ribbon Designer

You have two options:

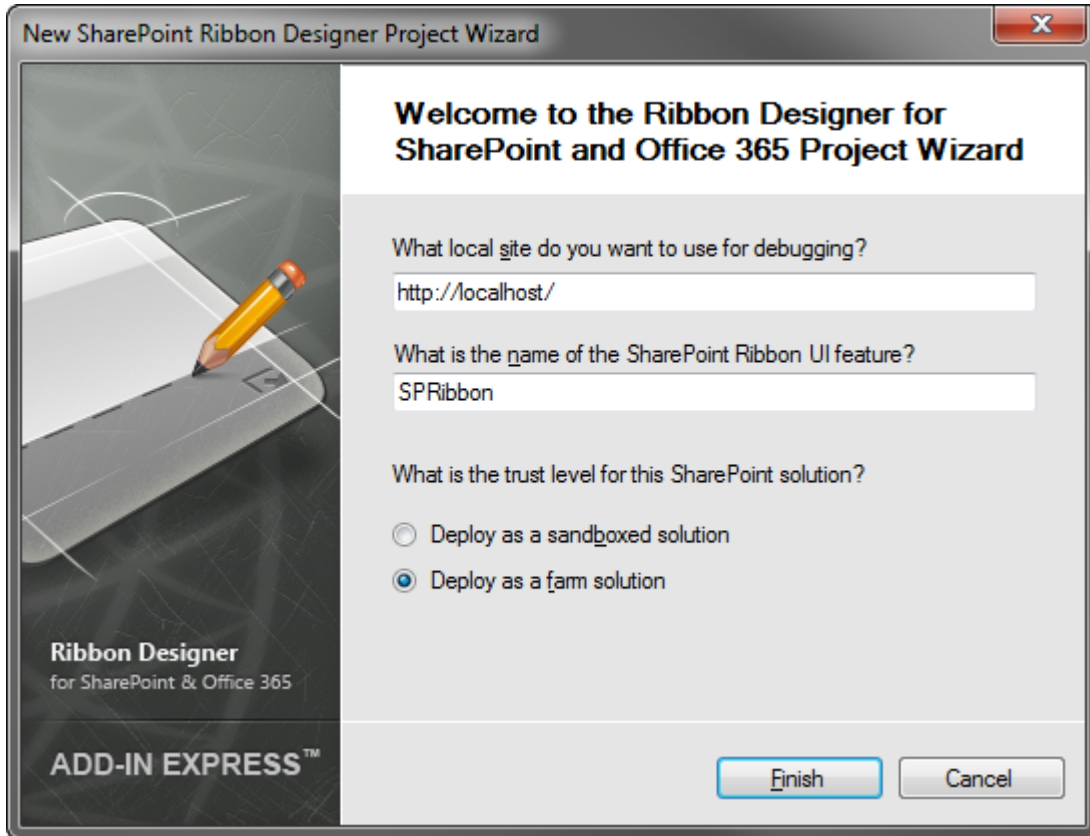
- [Create a new SharePoint project](#)
- [Add the Ribbon Designer to an existing SharePoint project](#)

Create a new SharePoint project:

In Visual Studio, open the *New Project* dialog and navigate to the *SharePoint* folder in the Visual Basic or Visual C# branch depending on your preferred programming language.



Select the *Add-in Express SharePoint Ribbon* item and click *OK*. This starts a one-step project wizard shown in the screenshot below:

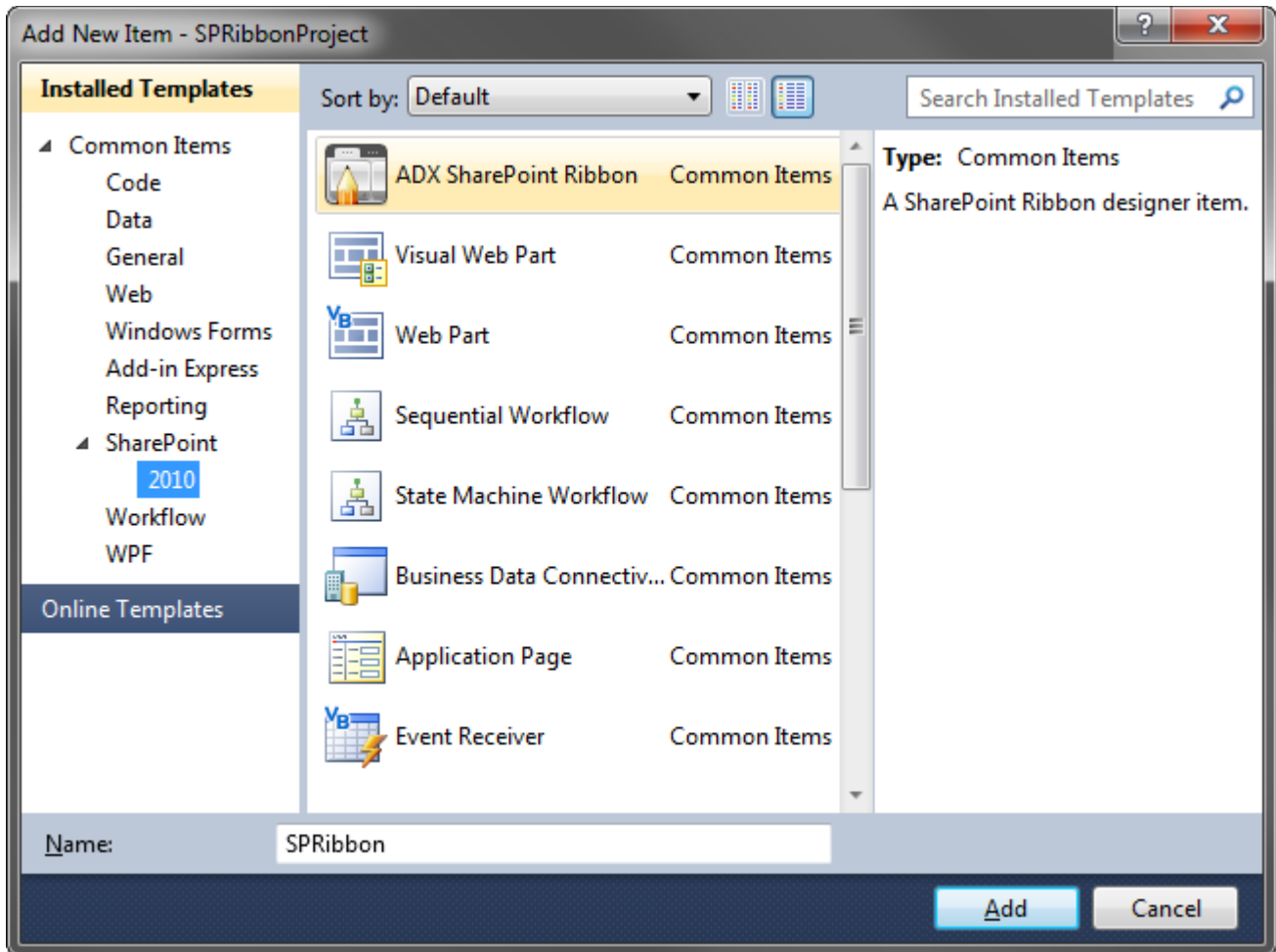


The project wizard requires that you specify a local site for debugging your solution. It allows you to choose the name of the feature enclosing the Ribbon Designer's functionality. Also, you must specify whether the solution will deploy to a server farm (default) or it will be a sandboxed solution. Clicking *Finish* creates and opens a new solution in the IDE. The solution contains just one project, a SharePoint project. We discuss it in [Ribbon Designer Basics](#).

See also: [Differences Between Sandboxed and Farm Solutions](#) and [Sandboxed Solution Considerations](#).

Add the Ribbon Designer to an existing SharePoint project:

Click the project's node in Solution Explorer. Then, on the *Project* menu, click *Add New Item* to open the *Add New Item* dialog. In the dialog, in the *Installed Templates* list, expand the *SharePoint* node, and then click *2010*.

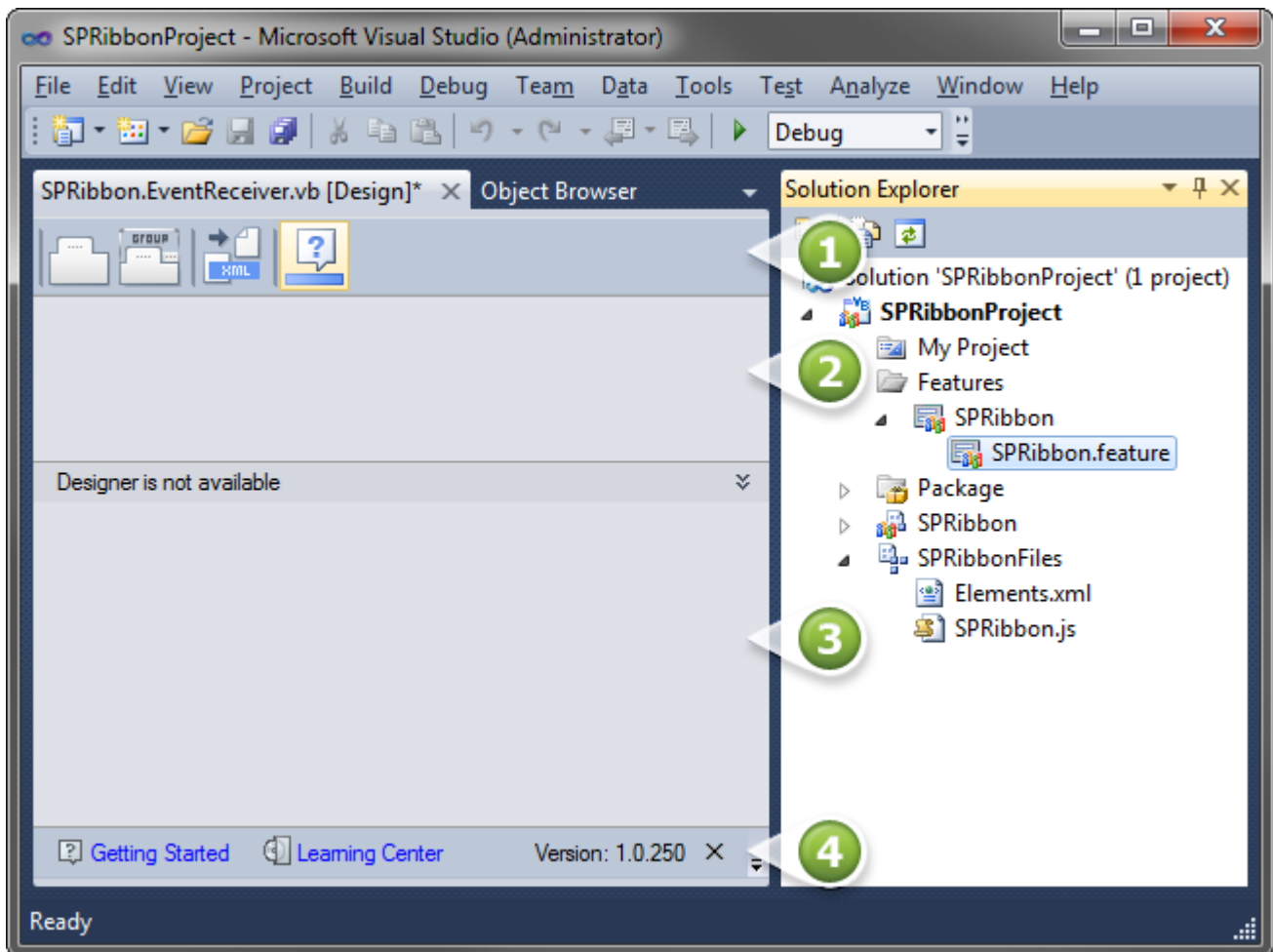


In the dialog, select *ADX SharePoint Ribbon*, enter a name in the *Name* text box and click *Add*.

Ribbon Designer Basics

Following the instructions above opens the Ribbon Designer in your project. The designer provides access to four areas:

- **Ribbon Toolbox** – (#1 in the screenshot below) it contains commands; clicking a command adds a corresponding Ribbon component to the designer surface.
- **Ribbon designer** - (#2 in the screenshot below) it is a usual designer.
- **In-place designer** - (#3 in the screenshot below) if there's a visual designer for the currently selected Add-in Express component, then it is shown in this area.
- **Help panel** – see #4 in the screenshot below.



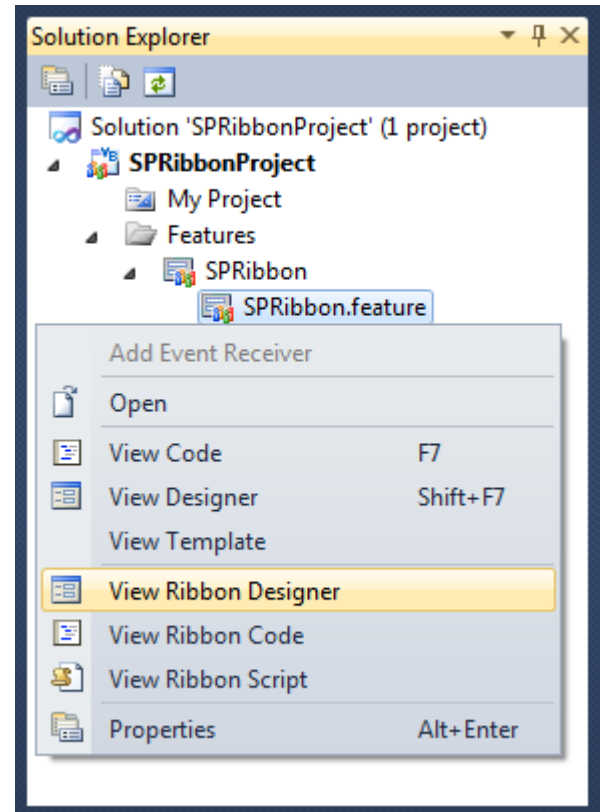
Before we start looking at how to use these areas, let us find the Ribbon Designer files in your project.

Ribbon Designer Files

The Ribbon Designer is a *feature* (in SharePoint Foundation terms). You specify the feature's name in the *Add New Item* dialog (see [Adding the Ribbon Designer](#)).

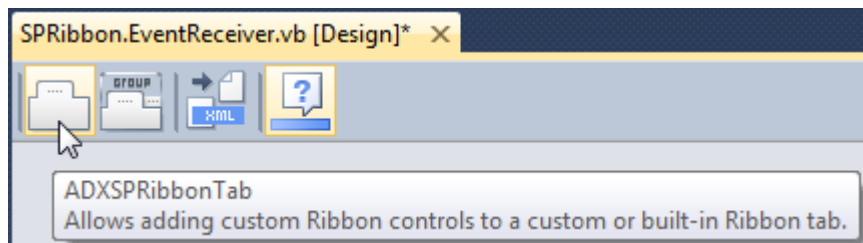
The main file is called `{feature name}.feature`. In this sample, we name the feature *SPRibbon*, the resulting file name is *SPRibbon.feature* (see the screenshot).

The feature provides three sections: designer, code and JavaScript code. To access them, you use the corresponding items of the context menu – just right-click the feature as shown in the screenshot.

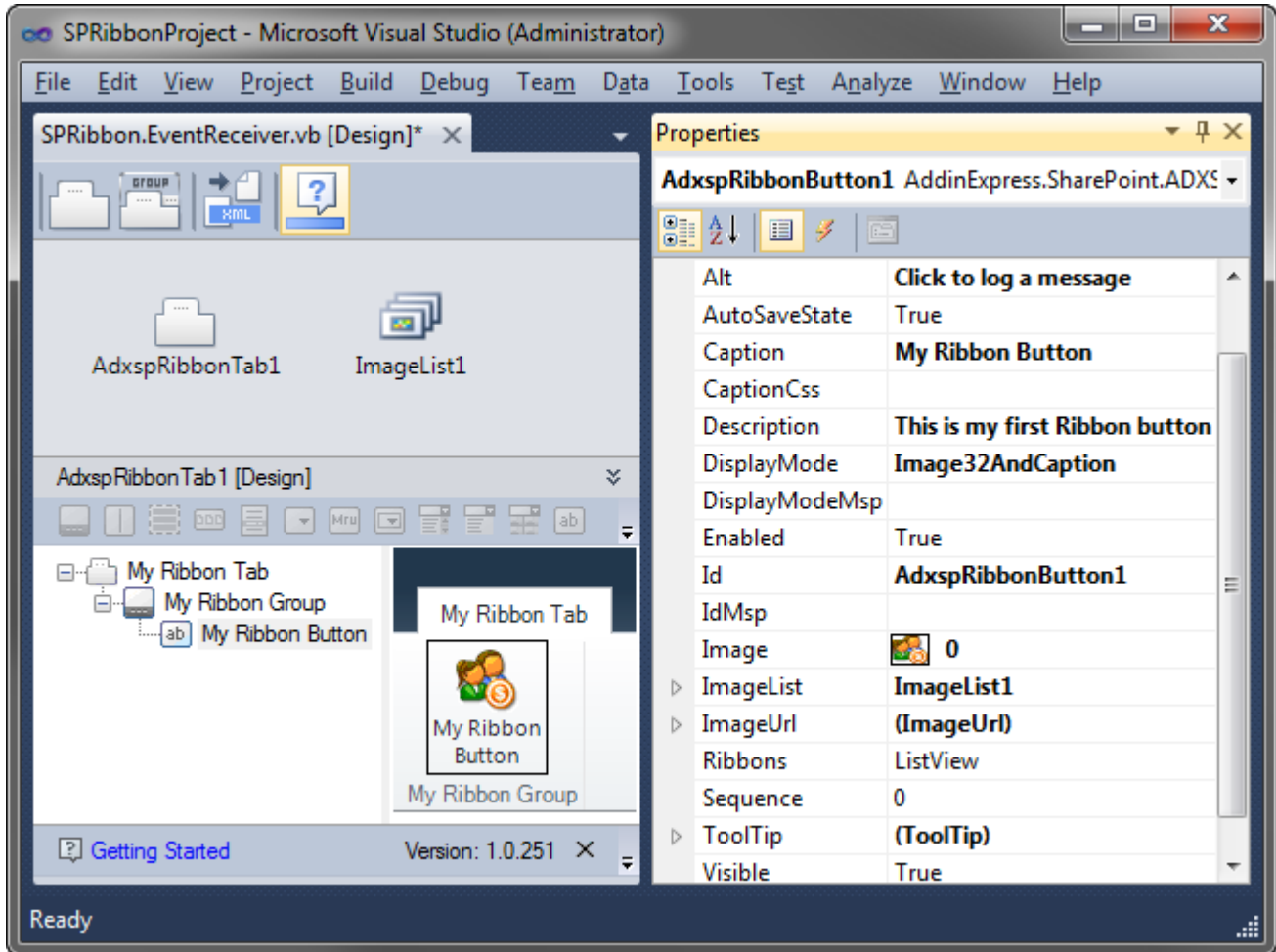


Ribbon Tabs and Controls

To add a new tab to the Ribbon, you use the `ADXSPRibbonTab` command on the Ribbon Toolbox to add an `AddinExpress.SharePoint.ADXSPRibbonTab` component to the designer.



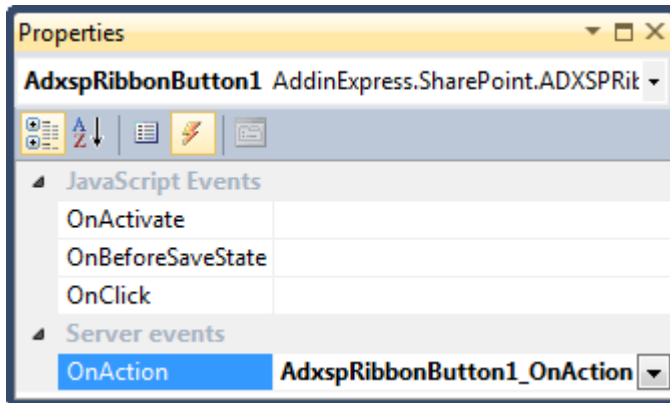
Then, in the in-place visual designer, use tool buttons to add or delete the Ribbon components that form the Ribbon interface of your SharePoint project.



To create the Ribbon UI shown in the screenshot above, change the caption of your tab to "My Ribbon Tab". Then, add a Ribbon group, and change its caption to "My Ribbon Group". Finally, select the group, add a button and set its caption to "My Ribbon Button". Use the *ImageList* and *Image* properties to set the image for the button; add an *ImageList* component onto the designer surface to provide images for your Ribbons. See also [Ribbon Controls in Detail](#), [Dealing with Built-in Ribbon Controls](#).

Server-side Events

Add an event handler to the *OnAction* event of the button and write the following code:



```
Private Sub AdxspRibbonButton1_OnAction(sender As System.Object, _
    e As AddinExpress.SharePoint.ADXSPButtonActionEventArgs) _
    Handles AdxspRibbonButton1.OnAction
    ' This event is not supported in sandboxed solutions.
    e.Result = ProcessMessage("Button is clicked")
End Sub
```

```
Private Function ProcessMessage(Message As String) As String
    Dim result As String = "Message is NOT processed"
    Dim rnd As New System.Random
    Dim val As Integer = rnd.Next()
    If CBool(val Mod 2) Then result = "Message is processed"
    Return "Message is:" + Message + " Result is:" + result
End Function
```

```
private void adxspRibbonButton1_OnAction(object sender,
    ADXSPButtonActionEventArgs e)
{
    // This event is not supported in sandboxed solutions.
    e.Result = ProcessMessage("Button is clicked");
}

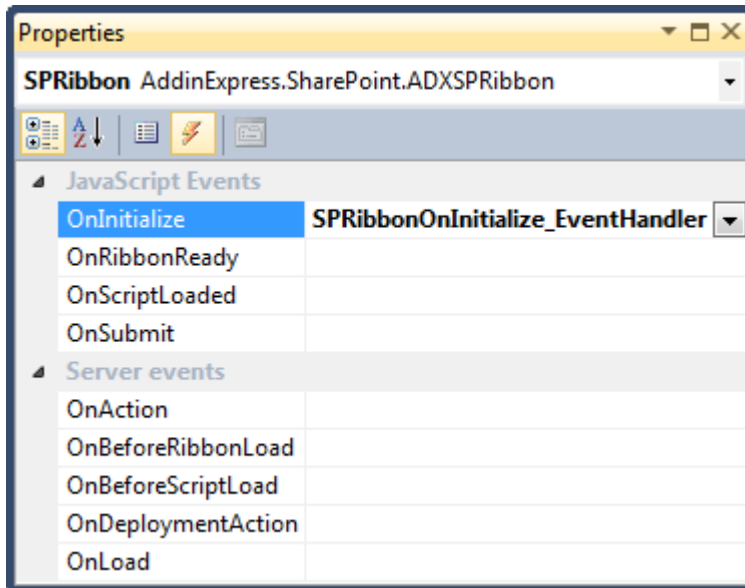
private string ProcessMessage(string Message)
{
    string result = "Message is NOT processed";
    System.Random rnd = new System.Random();
    int val = rnd.Next();
    if (val % 2 == 0)
```

```

    result = "Message is processed";
    return "Message is:" + Message + " Result is:" + result;
}

```

To handle the result, you add an event handler to the *OnInitialize* event (this is a JavaScript event) of the Ribbon Designer and write this code:



```

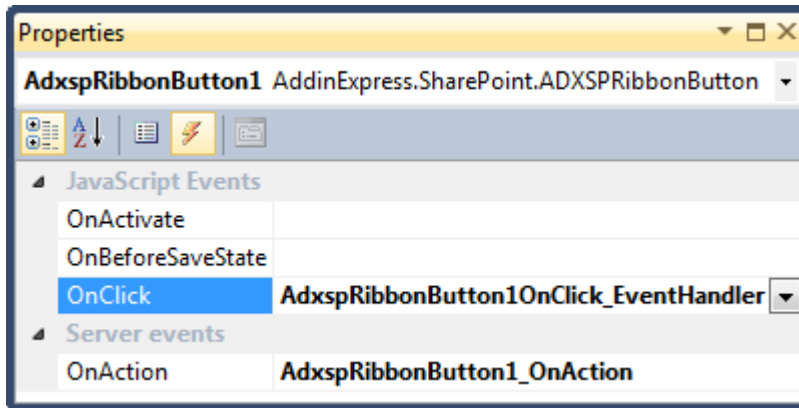
SPRibbonProject.SPRibbon.prototype.SPRibbonOnInitialize_EventHandler = function
(sender, isPostBack) {
    if (isPostBack) {
        var result = this.getPostBackResult();
        if (result != null && result != "") {
            alert(result);
        }
    }
}

```

Client-side Events

Now, let's handle an event of a Ribbon control on the client side.

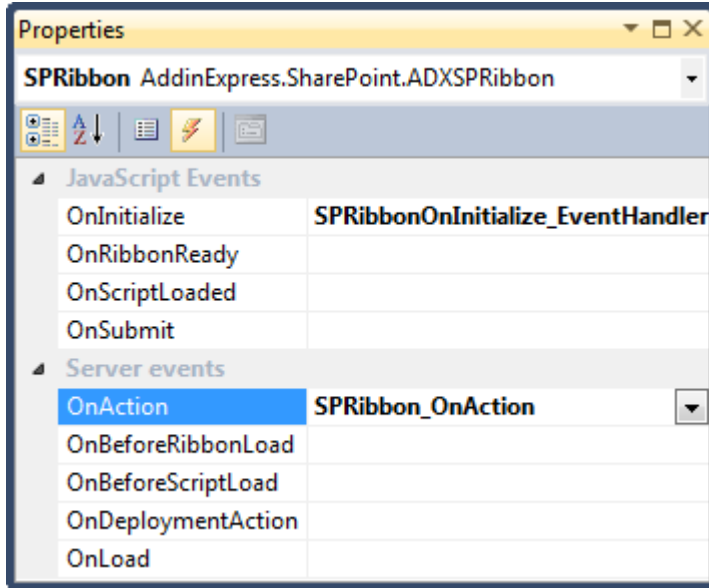
You start with adding an event handler to the *OnClick* event of the Ribbon button created on the previous step. Now, write this JavaScript code:



```
SPRibbonProject.SPRibbon.prototype.AdxspRibbonButton1OnClick_EventHandler =
function (sender) {
    var defaultMessage = "[no message]";
    var message = prompt("Enter a message", defaultMessage);
    if (message != null && message != "" & message != defaultMessage) {
        // See the OnAction event of the Ribbon Designer
        this.Submit("ProcessMessage", message);
    }
    else
        sender.AllowPostBack = false;
}
```

That is, when required conditions are met, the event handler calls the *Submit* method. The parameters passed to the method are the "ProcessMessage" string (which is interpreted as an action's ID) and the message itself. If conditions are **not** met, the event handler prevents generation of a postback request.

If the event handler for the *OnAction* event of the Ribbon Designer is supplied with the corresponding action ID, it invokes the *ProcessMessage* method as follows:



```
Private Sub SPRibbon_OnAction(sender As System.Object, _
    e As AddinExpress.SharePoint.ADXSPRibbonActionEventArgs) _
    Handles MyBase.OnAction
    ' This event is not supported in sandboxed solutions.
    If (e.ActionId = "ProcessMessage") Then
        e.Result = ProcessMessage(e.CustomData)
    End If
End Sub
```

```
Private Function ProcessMessage(Message As String) As String
    Dim result As String = "Message is NOT processed"
    Dim rnd As New System.Random
    Dim val As Integer = rnd.Next()
    If CBool(val Mod 2) Then result = "Message is processed"
    Return "Message is:" + Message + " Result is:" + result
End Function
```

```
private void SPRibbon_OnAction(object sender, ADXSPRibbonActionEventArgs e)
{
    // This event is not supported in sandboxed solutions.
    if (e.ActionId == "ProcessMessage")
        e.Result = ProcessMessage(e.CustomData);
}

private string ProcessMessage(string Message)
{
    string result = "Message is NOT processed";
```

```
System.Random rnd = new System.Random();
int val = rnd.Next();
if (val % 2 == 0)
    result = "Message is processed";
return "Message is:" + Message + " Result is:" + result;
}
```

Debugging the Project

For instructions on debugging SharePoint projects, see [Debugging SharePoint Solutions](#).

Deploying the Ribbon UI

For instructions on deploying SharePoint projects, see [Packaging and Deploying SharePoint Solutions](#). The functionality of the Ribbon Designer is provided in *AddinExpress.SharePoint.Ribbon.dll*; you must deploy this file onto the target server.

Deploying for Office 365

The Ribbon Designer completely supports sandboxed solutions, so it can be deployed on an Office 365 web site.

What's next?

We suggest that you check [Using the Ribbon Designer](#) below.

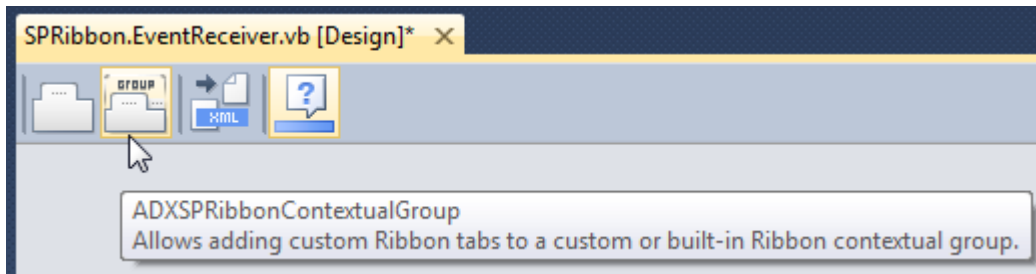
Using the Ribbon Designer

What are your options? Here are the answers...

Ribbon Controls in Detail

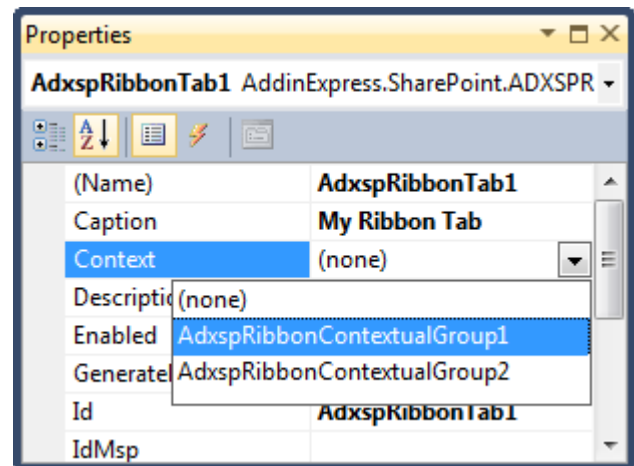
Creating and Showing Ribbon Contextual Groups

In the Add-in Express toolbox (see [Ribbon Designer Basics](#)), click the corresponding tool button to add an *ADXSPRibbonContextualGroup* component onto the designer surface.



You bind a Ribbon tab to a given contextual group using the tab's *Context* property as shown in the screenshot to the right.

To bind a contextual group to your Visual Web Part, you modify the *CreateChildControls* method so that it calls the *ShowContextualGroup* method. This is a [static](#) (Shared in VB.NET) method. We demonstrate how to use it in the code below:



```
Protected Overrides Sub CreateChildControls()
    Dim control As Control = Page.LoadControl(_ascxPath)
    Controls.Add(control)
    SPRibbon.ShowContextualGroup(Me.Page, GetType(SPRibbonProject.SPRibbon), _
        "AdxspRibbonContextualGroup1")
End Sub
```

In the code above, the *ShowContextualGroup* method specifies that the contextual group with the ID equal to "AdxspRibbonContextualGroup1" is shown for the web part.

Ribbon Control Visibility Rules

For a custom Ribbon control to be visible on a page, **all** the conditions below **must be** met:

- The *Visible* property of the control is **true** and the *ItemType* and *Ribbons* properties of the control correspond to the current context.
- If the direct parent of the control is a built-in control (see [Dealing with Built-in Ribbon Controls](#)), the control's *DisplayModeMsp* property must be set to a valid value (see [How Your Control is Displayed](#)).
- All container controls (including the group and tab controls) in the hierarchy of which the control is located are visible; in other words, the visibility and context matching requirement above apply to the container controls, too.
- If the ribbon tab to which the control belongs is bound to a contextual group (see [Creating and Showing Ribbon Contextual Groups](#)), then the group must be visible, too; if the group is bound to a web part, then the web part must be added to the current page.

How Your Control is Displayed

For a custom control located **on a custom container control** (such as custom group), you use the *DisplayMode* property. Say, for a custom Ribbon button, you set the *DisplayMode* property to one of the following values:

- *Image16Only* – a 16x16 image (.PNG, .ICO or .BMP)
- *Image16AndCaption* – a 16x16 image (.PNG, .ICO or .BMP) plus caption
- *Image32AndCaption* – a 32x32 image (.PNG, .ICO or .BMP) plus caption

Using any of these values implies that you also specify an image using the *Image* and *ImageList* properties.

If, however, the custom control is located **on a built-in container control**, you must set the *DisplayModeMsp* property of the custom control; the property editor suggests values appropriate for the built-in container. If you don't set a value to this property, your custom control located on a built-in container control will not show up, pay attention to [Ribbon Control Visibility Rules](#) and [Dealing with Built-in Ribbon Controls](#).

Ribbon Designer Programmability

Localization

The Ribbon Designer supports localization by providing the *Language* and *Localizable* properties: click the designer surface and look at the *Properties* window. See also [Localizing SharePoint Solutions](#).

Accessing Ribbon Controls in JavaScript

If you have a text box with its *ID* set to "AdxspRibbonTextBox1", then you can write the code below in the *OnClick* event of a Ribbon button:

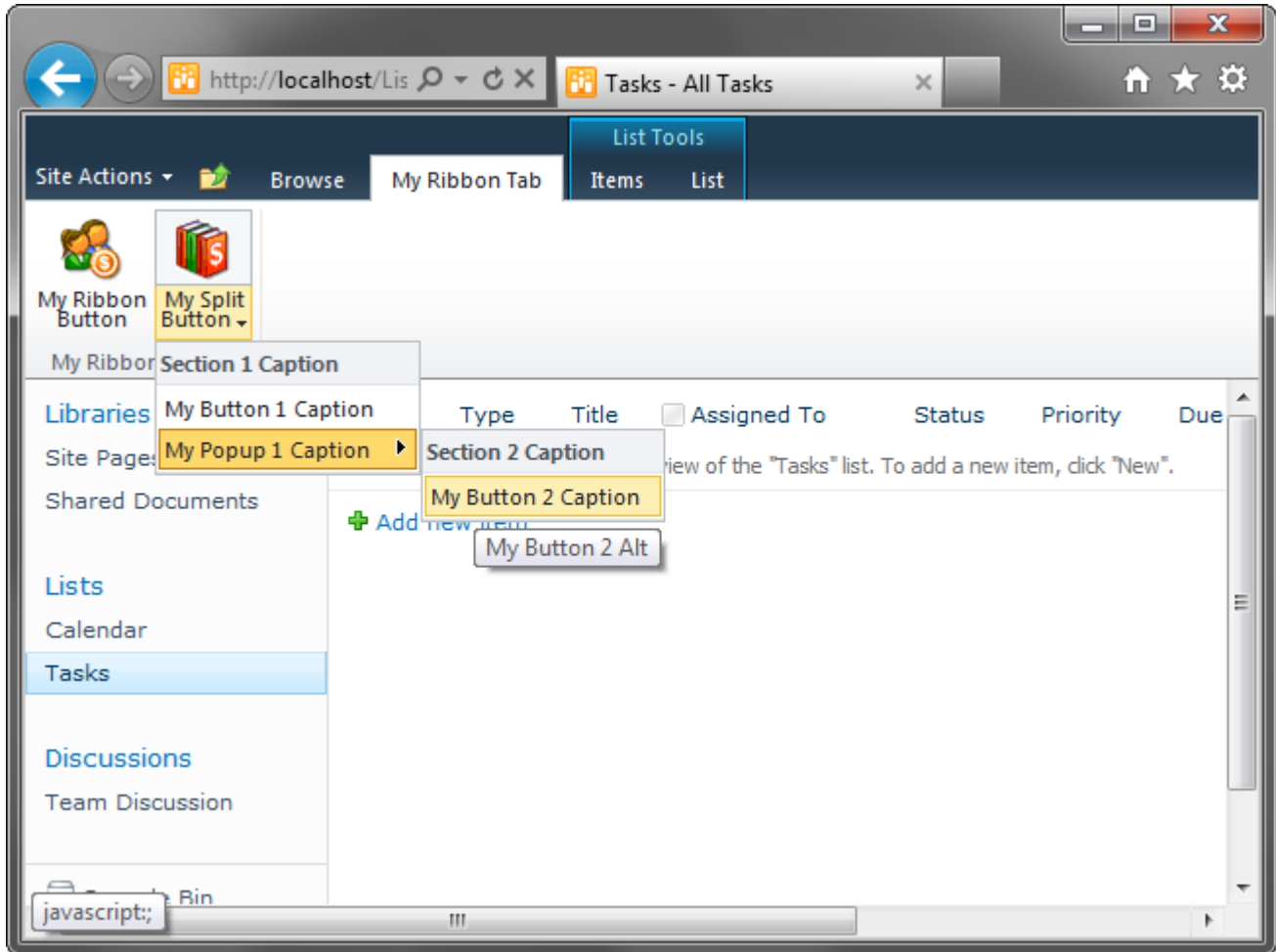
```
SPRibbonProject.SPRibbon.prototype.AdxspRibbonButton2OnClick_EventHandler =  
function (sender) {  
    var textBox = this.FindControl("AdxspRibbonTextBox1");  
    var isOk = false;  
    if (textBox != null) {  
        var text = textBox.Text;  
        if (text != null && text != "") {  
            isOk = true;  
            alert(text);  
        }  
    }  
    if (!isOk) alert("Something went wrong or no text was supplied");  
    sender.AllowPostBack = false;  
}
```

That is, in the event handler above, you find the text box using the *FindControl* method; it accepts the ID of a control to find. Then the text is extracted and an alert is shown.

When testing the method above, you'll find that `FindControl()` returns the previous value of the text box unless you press {Enter} when entering text. Alas, this is how the Ribbon text box works and you can do nothing about this.

Creating Ribbon Controls Dynamically

For Ribbon controls containing a menu (e.g. *ADXSPRibbonSplitButton*), you use the property *PopulateDynamically* to populate the menu dynamically. Setting this property to *True* results in generating the *OnMenuCreate* event on the corresponding object of the Ribbon Designer JavaScript client object model. You handle this event to create Ribbon controls on the client side.



In the screenshot above, the split button is created at design time as described in [Getting Started](#). The controls residing on the menu part of the split button are created with this code:

```
SPRibbonProject.SPRibbon.prototype.CreateMenu = function (sender) {
    // Section 1
    var menuSection = sender.AddMenuSection("MySection1ID");
    menuSection.Caption = "Section 1 Caption";
    // menuSection.DisplayMode = "Image16AndCaption";

    // Button 1
    var button = menuSection.AddControl("MyButton1ID", "Button");
    button.Description = "My Button 1 Description.";
    button.Caption = "My Button 1 Caption";
    // button.Alt = "My Button 1 Alt.";
    // button.Enabled = true;
    // button.ToolTipTitle = "My Button 1 Tooltip Title";
    // button.ToolTipDescription = "My Button 1 Tooltip Description";
    // button.ToolTipHelpKeyword = "My Button 1";
    // button.ToolTipShortcutKey = "Ctrl+Shift+Z";
}
```

```

button.OnClickEventName = "ShowId";

var popup = menuSection.AddControl("MyPopup1ID", "FlyoutAnchor");
popup.PopulateDynamically = false;
popup.Caption = "My Popup 1 Caption";
//   popup.OnMenuCreateEventName = "OnPopulate_DynamicHandler";

var menuSection2 = popup.AddMenuSection("MySection2ID ");
menuSection2.Caption = "Section 2 Caption";

var button2 = menuSection2.AddControl("MyButton2ID", "Button");
button2.Description = "My Button 2 Description.";
button2.Caption = "My Button 2 Caption";
button2.Alt = "My Button 2 Alt";
button2.OnClickEventName = "ShowId";
}

SPRibbonProject.SPRibbon.prototype.ShowId = function (sender) {
    alert(sender.getId());
}

```

Preserving the State of Ribbon Controls

All components representing Ribbon controls allow retaining the controls' state. Both *ADXSPRibbon* and *ADXSPPageRibbon* classes have the *AutoSaveMode* property that you can set to *Local*, *Session* (default) or *None*. All Ribbon components provide the *AutoSaveState* property accepting *True* (default) or *False*. You can specify whether to preserve the state of a given Ribbon control at runtime: just set the *AutoSave* property to *False* in the JavaScript event *OnBeforeSaveState* that every Ribbon component provides.

Ribbon Designer Events

The *ADXSPRibbon* class provides events listed below.

Event name	Event description
<i>OnAction</i>	Occurs when the designer class is invoked via the Submit method of the Ribbon Designer client object model. Allows handling custom actions and passing data to them. A sample of use is given in Client-side Events . Not supported in sandboxed solutions.
<i>OnBeforeRibbonLoad</i>	Occurs before the Ribbon XML is supplied to the Server Ribbon. Allows customizing the Ribbon XML at run-time. Not supported in sandboxed solutions.
<i>OnBeforeScriptLoad</i>	Occurs before the Ribbon Designer client library is registered for the Server Ribbon. Allows customizing the Ribbon Designer script at run-time. Not supported in sandboxed solutions.

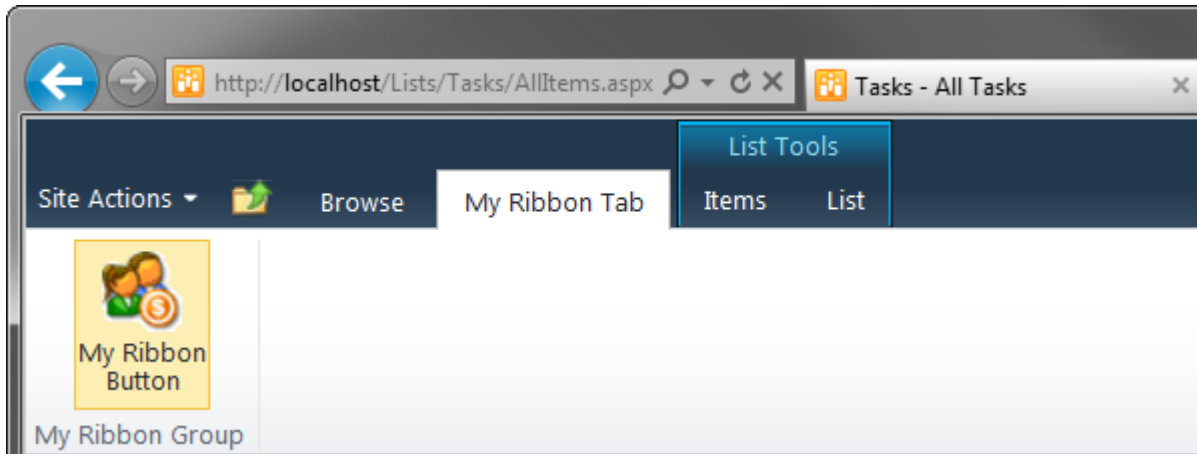
<i>OnDeploymentAction</i>	Occurs when the designer feature is installed, uninstalled, activated, deactivated, or upgraded.
<i>OnInitialize</i>	JavaScript. Occurs when the Ribbon Designer client library is loaded by the browser. You can use this event to initialize the state of Ribbon controls. A sample of use is given in Server-side Events .
<i>OnLoad</i>	Occurs when the Ribbon Designer is loaded into the <i>Page</i> object. Not supported in sandboxed solutions.
<i>OnRibbonReady</i>	JavaScript. Occurs when the ribbon control is constructed. You can use this event to activate a Ribbon tab (<i>SelectTabById</i>) or check if a tab is available (<i>IsTabAvailable</i>).
<i>OnScriptLoaded</i>	JavaScript. Occurs when the Ribbon Designer script is loaded into the Page. You can use this event to perform actions depending on whether your script is loaded.
<i>OnSubmit</i>	JavaScript. Occurs when the FORM is about to be submitted.

Dealing with Built-in Ribbon Controls

To capture the images below, we used the controls described in the [Ribbon Tabs and Controls](#) section. Here we will modify their properties one by one to give answers to typical questions.

Add a custom tab

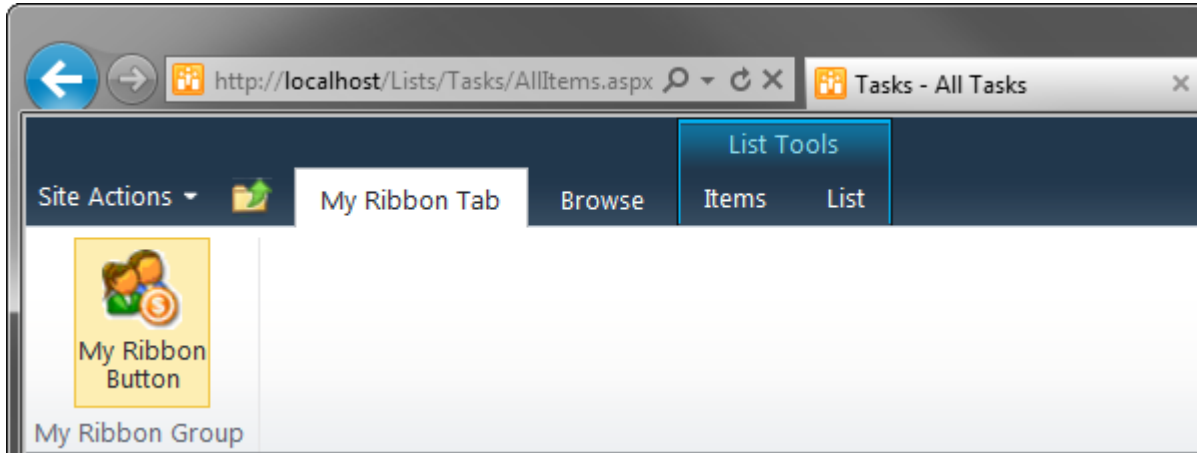
Create a tab and specify the Ribbons and item types for which you show the tab.



<i>Property</i>	<i>Value</i>
ADXSPRibbonTab.ItemType	[Task]
ADXSPRibbonTab.Ribbons	[ListView]

Position a custom tab

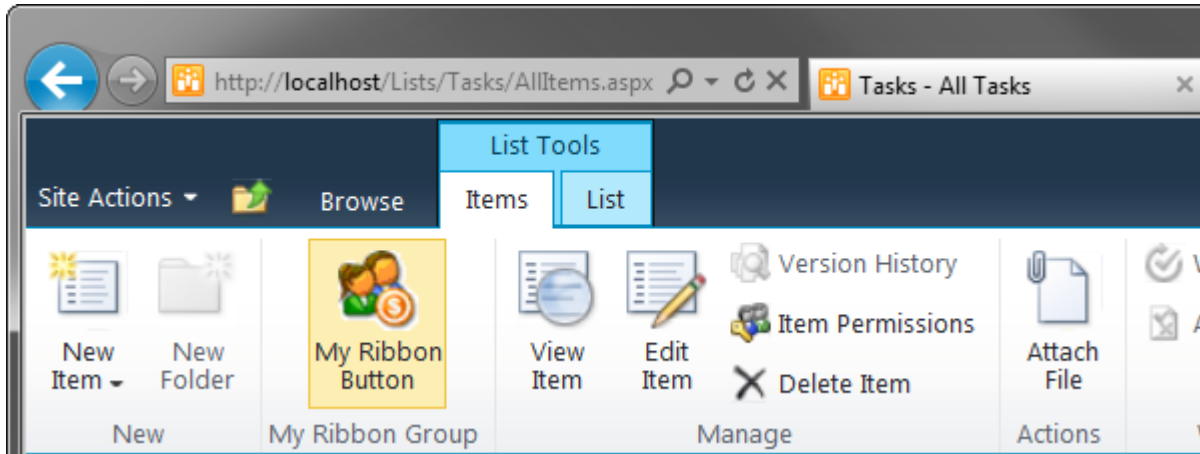
Set the *Sequence* property to specify the position of your tab among other tabs.



<i>Property</i>	<i>Value</i>
ADXSPRibbonTab.ItemType	[Task]
ADXSPRibbonTab.Ribbons	[ListView]
ADXSPRibbonTab.Sequence	10

Add a custom group to a built-in tab

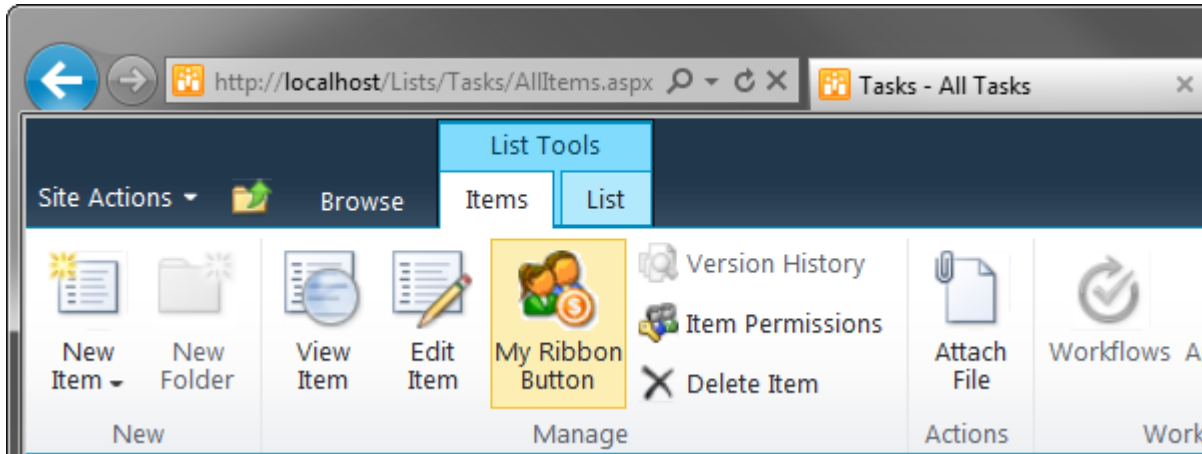
Specify the ID of the built-in tab to which you add your group and set the *Sequence* property to position your group among existing groups.



<i>Property</i>	<i>Value</i>
ADXSPRibbonTab.IdMsp	[ListItem]
ADXSPRibbonTab.ItemType	[Task]
ADXSPRibbonTab.Ribbons	[ListView]
ADXSPRibbonGroup.Sequence	15

Add a custom control to a built-in group

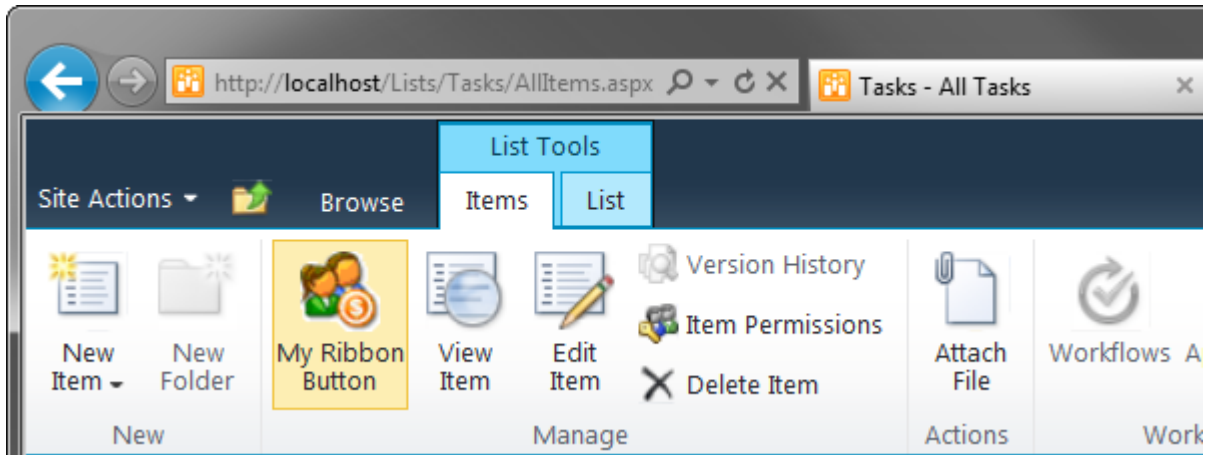
Specify the ID of the built-in group to which you add your control. Set the *DisplayModeMsp* property to specify the way in which your control will be shown. See also [How Your Control is Displayed](#).



<i>Property</i>	<i>Value</i>
ADXSPRibbonTab.IdMsp	[ListItem]
ADXSPRibbonTab.ItemType	[Task]
ADXSPRibbonTab.Ribbon	[ListView]
ADXSPRibbonGroup.IdMsp	[ListItem.Manage]
ADXSPRibbonButton.DisplayModeMsp	[o1]

Position a custom control in a built-in group

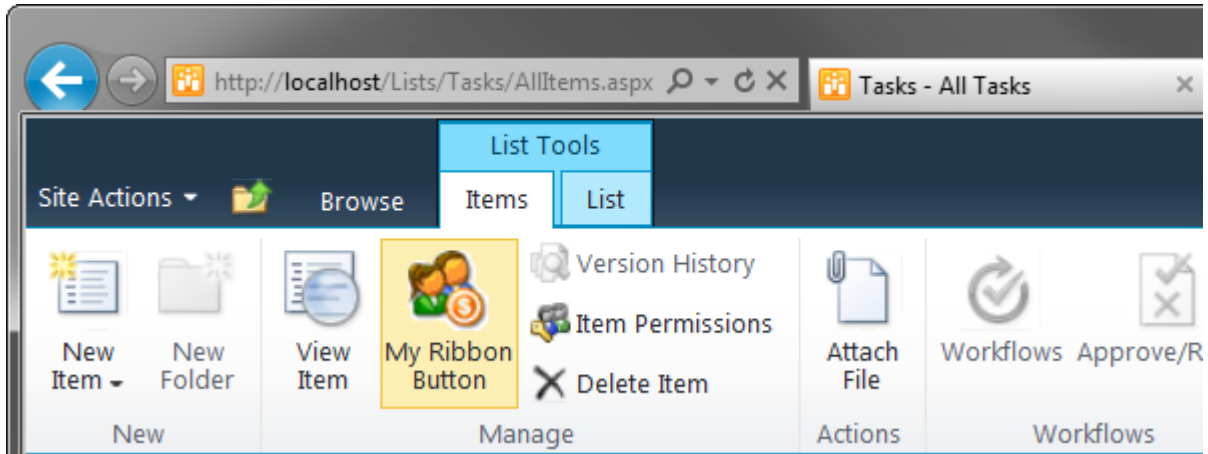
Set the *Sequence* property to specify the position of your control among other controls.



<i>Property</i>	<i>Value</i>
ADXSPRibbonTab.IdMsp	[ListItem]
ADXSPRibbonTab.ItemType	[Task]
ADXSPRibbonTab.Ribbons	[ListView]
ADXSPRibbonGroup.IdMsp	[ListItem.Manage]
ADXSPRibbonButton.DisplayModeMsp	[o1]
ADXSPRibbonButton.Sequence	1

Customize a built-in button

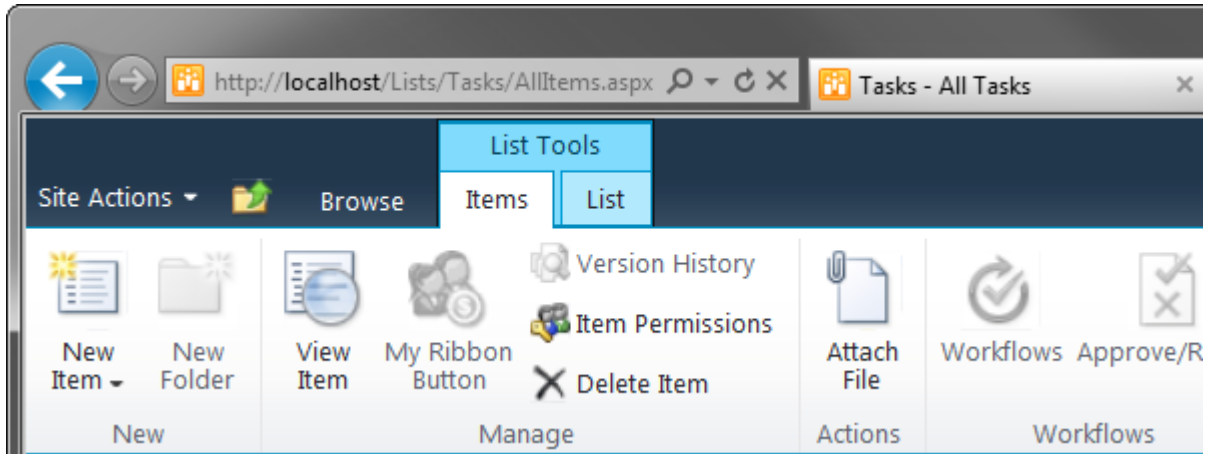
Specify the ID of the built-in control to override. Note that the built-in control isn't customized – it is replaced with a custom control.



<i>Property</i>	<i>Value</i>
ADXSPRibbonTab.IdMsp	[ListItem]
ADXSPRibbonTab.ItemType	[Task]
ADXSPRibbonTab.Ribbons	[ListView]
ADXSPRibbonGroup.IdMsp	[ListItem.Manage]
ADXSPRibbonButton.DisplayModeMsp	[o1]
ADXSPRibbonButton.IdMsp	[ListItem.Manage.EditProperties]

Disable a built-in control

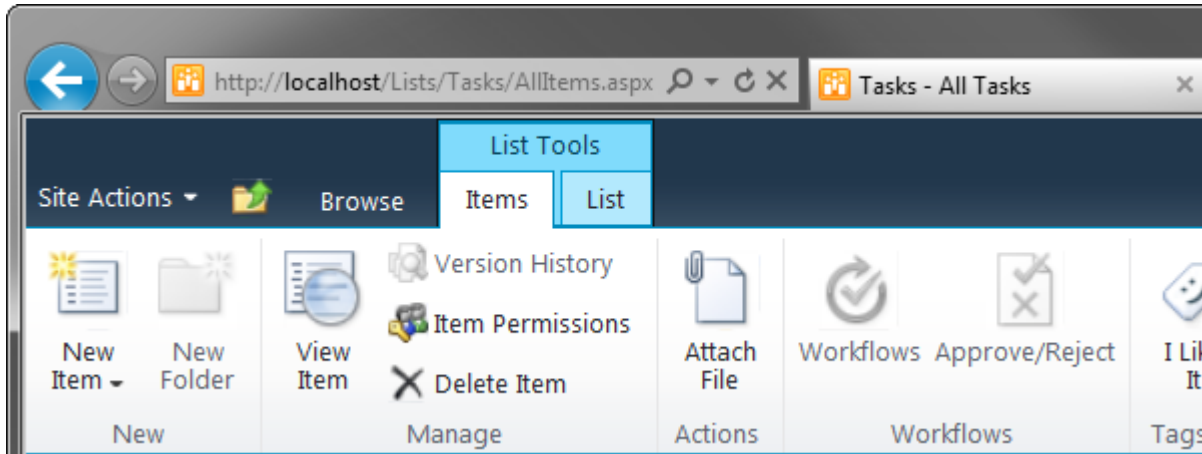
The Server Ribbon schema doesn't allow disabling a built-in control. You can replace the built-in control with a custom one (see [Customize a built-in button](#)) and set `Enabled = false` for the custom control.



<i>Property</i>	<i>Value</i>
ADXSPRibbonTab.IdMsp	[ListItem]
ADXSPRibbonTab.ItemType	[Task]
ADXSPRibbonTab.Ribbons	[ListView]
ADXSPRibbonGroup.IdMsp	[ListItem.Manage]
ADXSPRibbonButton.DisplayModeMsp	[o1]
ADXSPRibbonButton.IdMsp	[ListItem.Manage.EditProperties]
ADXSPRibbonButton.Enabled	false

Hide a built-in control

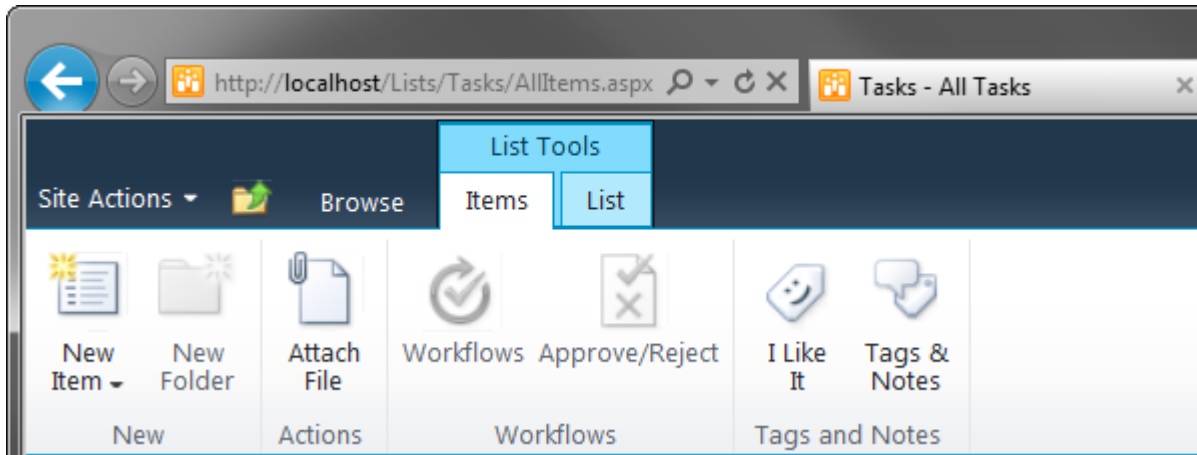
The current version of the Ribbon Designer doesn't provide an option to hide a built-in control directly. However, you can replace the built-in control with a custom one (see [Customize a built-in button](#)) and set `Visible = false` for the custom control.



<i>Property</i>	<i>Value</i>
ADXSPRibbonTab.IdMsp	[ListItem]
ADXSPRibbonTab.ItemType	[Task]
ADXSPRibbonTab.Ribbons	[ListView]
ADXSPRibbonGroup.IdMsp	[ListItem.Manage]
ADXSPRibbonButton.DisplayModeMsp	[o1]
ADXSPRibbonButton.IdMsp	[ListItem.Manage.EditProperties]
ADXSPRibbonButton.Visible	false

Hide a built-in group

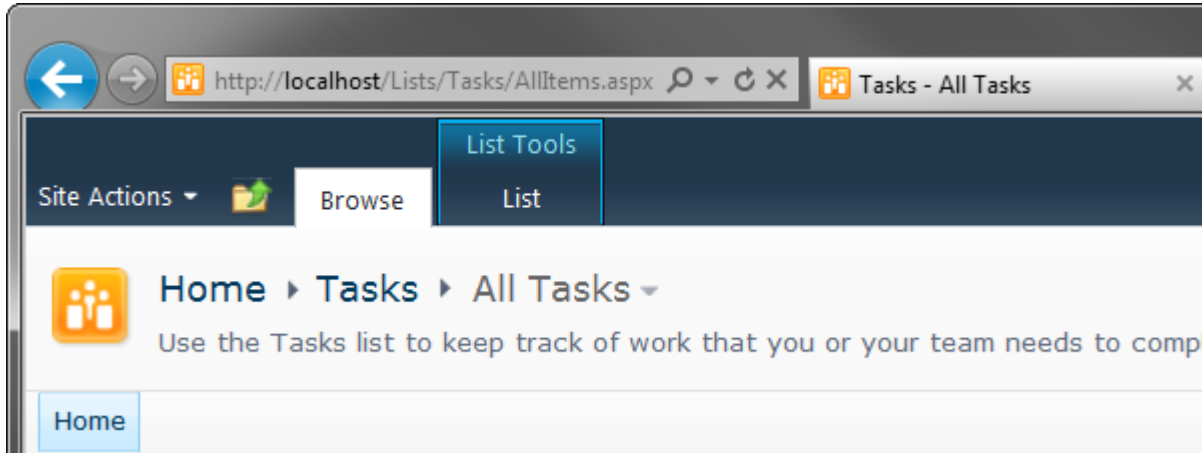
For the overridden built-in group set `Visible = false`.



<i>Property</i>	<i>Value</i>
ADXSPRibbonTab.IdMsp	[ListItem]
ADXSPRibbonTab.ItemType	[Task]
ADXSPRibbonTab.Ribbons	[ListView]
ADXSPRibbonGroup.IdMsp	[ListItem.Manage]
ADXSPRibbonGroup.Visible	false

Hide a built-in tab

For the overridden built-in tab set `Visible = false`.



<i>Property</i>	<i>Value</i>
ADXSPRibbonTab.IdMsp	[ListItem]
ADXSPRibbonTab.ItemType	[Task]
ADXSPRibbonTab.Ribbons	[ListView]
ADXSPRibbonTab.Visible	false

Use a built-in control on a custom control container

The Server Ribbon schema doesn't support adding a built-in control to a custom container control such as tab, group, etc.

Disable a built-in group or tab

The Server Ribbon schema doesn't allow disabling a built-in container control such as a tab or group. You may choose to hide the built-in container control, add a custom container control imitating the look of the built-in group or tab and disable it.

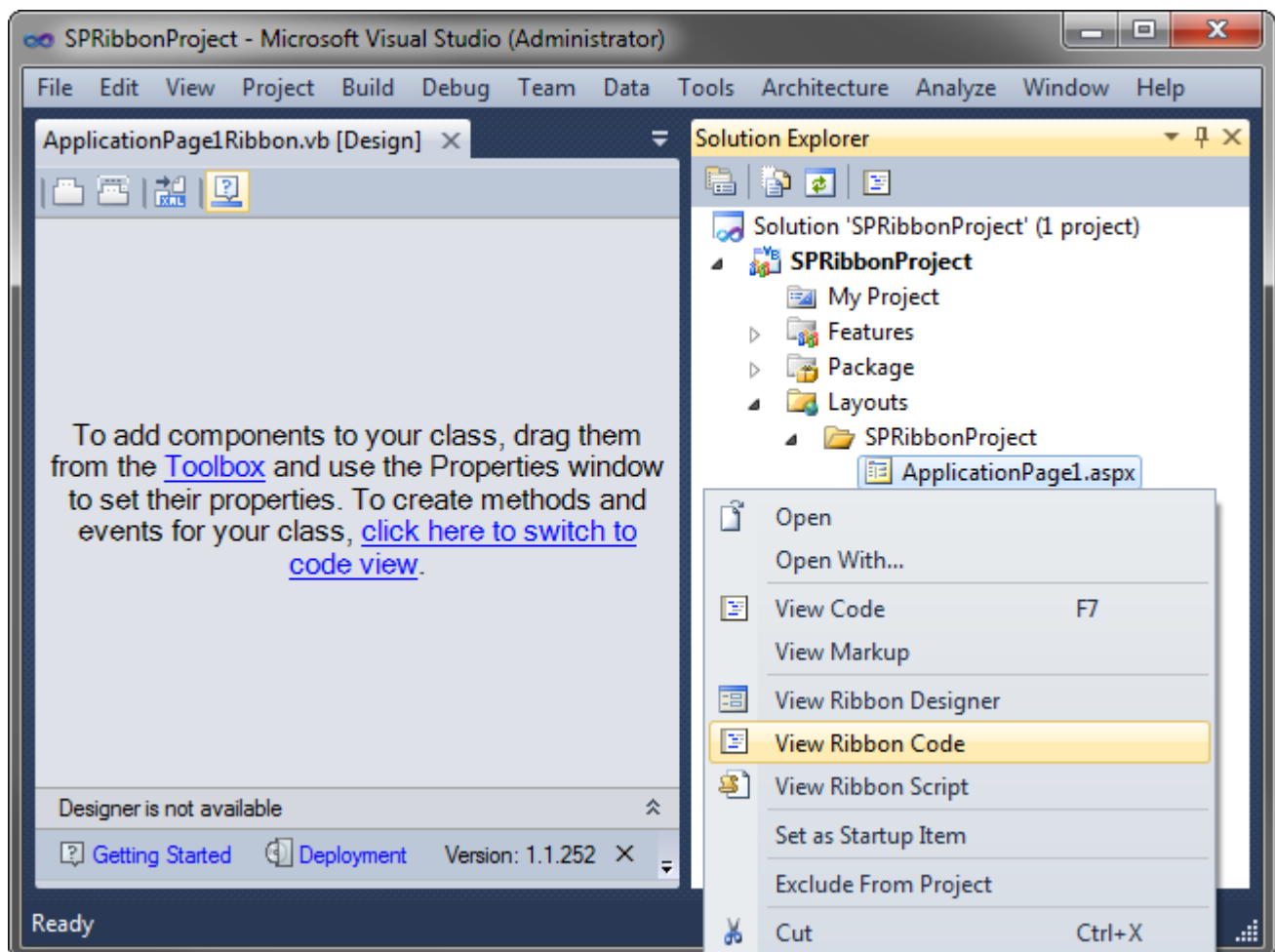
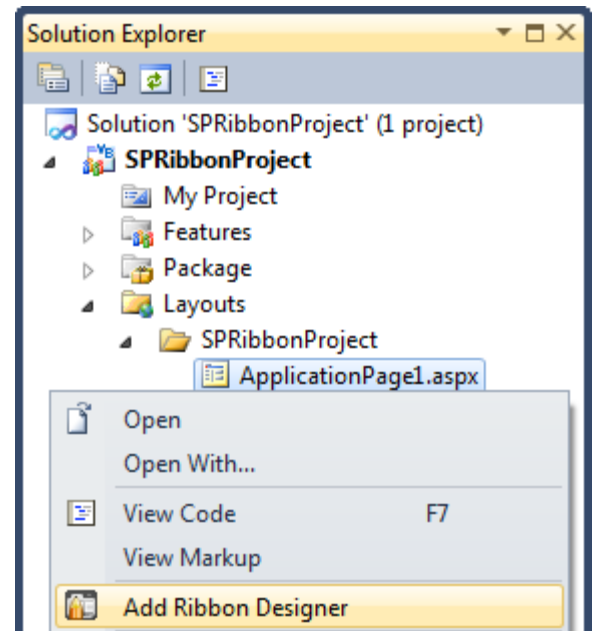
Integrating with Existing Solutions

The Ribbon UI and your SharePoint application

In a farm solution only, you can add the Ribbon Designer to an Application Page, Visual Web Part, or User Control. This section shows how to add the Ribbon Designer to an existing Application Page but the below also applies to adding the Ribbon Designer to a Visual Web Part or User Control.

So, you start with right-clicking *ApplicationPage1.aspx* in the Solution Explorer and choosing *Add Ribbon Designer* in the context menu.

Doing this adds the Ribbon Designer component to the Application Page. Now, you can access all sections of the Ribbon Designer using the same context menu.



The Ribbon Designer adds some initialization code to the code of the page. In this sample project, the resulting code in *ApplicationPage1.aspx* is shown below, pay attention to the code line in bold:

```
Imports System
Imports Microsoft.SharePoint
Imports Microsoft.SharePoint.WebControls

Namespace Layouts.SPRibbonProject

    Partial Public Class ApplicationPage1
        Inherits LayoutsPageBase

        Protected Overrides Sub CreateChildControls()
            ' Ribbon Designer for SharePoint and Office 365 generated code
            Me.Controls.Add(New ApplicationPage1RibbonDelegate())

            MyBase.CreateChildControls()
        End Sub

        Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
            Handles Me.Load

        End Sub

    End Class

End Namespace
```

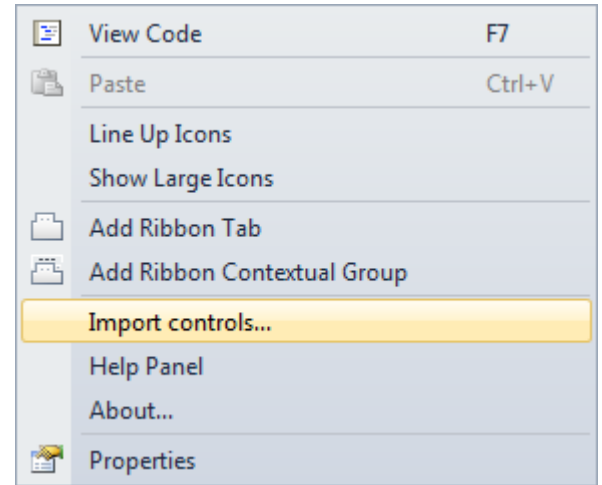
Note that in this environment the Ribbon Designer components do **not** provide some properties and events. For instance, you cannot use (and won't receive) the *OnDeploymentAction* event. Also, Ribbon control components e.g. *ADXSPRibbonButton* do **not** support some properties when added onto an Application Page, Visual Web Part or User Control; these properties are listed below:

- *Image*
- *ImageList*
- *Ribbons*

Importing Existing Ribbon Controls

Right-click the designer surface, choose Import Controls in the context menu and select an XML file containing Ribbon controls. The Ribbon Designer verifies the file and creates components corresponding to controls mentioned in the XML file. To be imported, the controls must be located in a hierarchy of Ribbon controls, the top control of the hierarchy must be a group, tab, or contextual tab.

See an example of the Server Ribbon XML in [Walkthrough: Adding a Tab to the Server Ribbon](#).



Finally

If you did not find answers to your questions in this manual, please contact our support team, see [Technical Support](#).