ektron

What do you want your website to do?

Ektron PageBuilder QuickStart Guide

For Users, Administrators, and Developers

Streamline the process of building new pages on your Web site

ektron CMS400.net

EKTRON, INC. SOFTWARE LICENSE AGREEMENT

YOUR RIGHT TO USE THE PRODUCT DELIVERED IS SUBJECT TO THE TERMS AND CONDITIONS SET OUT IN THIS LICENSE AGREEMENT. USING THIS PRODUCT SIGNIFIES YOUR AGREEMENT TO THESE TERMS. IF YOU DO NOT AGREE TO THIS SOFTWARE LICENSE AGREEMENT, DO NOT DOWNLOAD.

CUSTOMER should carefully read the following terms and conditions before using the software program(s) contained herein (the "Software"). Downloading and/or using the Software or copying the Software onto CUSTOMER'S computer hard drive indicates CUSTOMER'S acceptance of these terms and conditions. If CUSTOMER does not agree with the terms of this agreement, CUSTOMER should not download.

Ektron, Inc. ("Ektron") grants, and the CUSTOMER accepts, a nontransferable and nonexclusive License to use the Software on the following terms and conditions:

1. Right to use: The Software is licensed for use only in delivered code form. Each copy of the Software is licensed for use only on a single URL. Each license is valid for the number of seats listed below (the "Basic Package"). Any use of the Software beyond the number of authorized seats contained in the Basic Package without paying additional license fees as provided herein shall cause this license to terminate. Should CUSTOMER wish to add seats beyond the seats licensed in the Basic Package, the CUSTOMER may add seats on a block basis at the then current price for additional seats (see product pages for current price). The Basic Packages are as follows:

Ektron CMS400.NET — Licensed for ten seats (10 named users) per URL.

Ektron eWebEditPro — Licensed for ten seats (10 named users) per URL.

Ektron eWebEditPro+XML — Licensed for ten seats (10 named users) per URL.

For purposes of this section, the term "seat" shall mean an individual user provided access to the capabilities of the Software.

The CUSTOMER may not modify, alter, reverse engineer, disassemble, or decompile the Software. This software product is licensed, not sold.

2. Duration: This License shall continue so long as CUSTOMER uses the Software in compliance with this License. Should CUSTOMER breach any of its obligations hereunder, CUSTOMER agrees to return all copies of the Software and this License upon notification and demand by Ektron.

3. Copyright: The Software (including any images, "applets," photographs, animations, video, audio, music and text incorporated into the Software) as well as any accompanying written materials (the "Documentation") is owned by Ektron or its suppliers, is protected by United States copyright laws and international treaties, and contains confidential information and trade secrets. CUSTOMER agrees to protect the confidentiality of the Software and Documentation. CUSTOMER agrees that it will not provide a copy of this Software or Documentation nor divulge any proprietary information of Ektron to any person, other than its employees, without the prior consent of Ektron; CUSTOMER shall use its best efforts to see that any user of the Software licensed hereunder complies with this license.

4. Limited Warranty: Ektron warrants solely that the medium upon which the Software is delivered will be free from defects in material and workmanship under normal, proper and intended usage for a period of three (3) months from the date of receipt. Ektron does not warrant the use of the Software will be uninterrupted or error free, nor that program errors will be corrected. This limited warranty shall not apply to any error or failure resulting from (i) machine error, (ii) Customer's failure to follow operating instructions, (iii) negligence or accident, or (iv) modifications to the Software by any person or entity other than Company. In the event of a breach of warranty, Customer's sole and exclusive remedy, is repair of all or any portion of the Software. If such remedy fails of its essential purpose, Customer's sole remedy and Ektron's maximum liability shall be a refund of the paid purchase price for the defective Products only. This limited warranty is only valid if Ektron receives written notice of breach of warranty within thirty days after the warranty period expires.

5. Limitation of Warranties and Liability: THE SOFTWARE AND DOCUMENTATION ARE SOLD "AS IS" AND WITHOUT ANY WARRANTIES AS TO THE PERFORMANCE, MERCHANTIBILITY, DESIGN, OR OPERATION OF THE SOFTWARE. NO WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE IS OFFERED. EXCEPT AS DESCRIBED IN SECTION 4, ALL WARRANTIES EXPRESS AND IMPLIED ARE HEREBY DISCLAIMED.

NEITHER COMPANY NOR ITS SUPPLIERS SHALL BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS OR GOODWILL, LOSS OF DATA OR USE OF DATA, INTERRUPTION OF BUSINESS NOR FOR ANY OTHER INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND UNDER OR ARISING OUT OF, OR IN ANY RELATED TO THIS AGREEMENT, HOWEVER, CAUSED, WHETHER FOR BREACH OF WARRANTY, BREACH OR REPUDIATION OF CONTRACT, TORT, NEGLIGENCE, OR OTHERWISE, EVEN IF COMPANY OR ITS REPRESENTATIVES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSS.

6. Additional Terms and Conditions apply

When using the CMS400 map control, Subject to the terms and conditions of the Map provider (Microsoft Virtual Earth or Google maps)

Microsoft Virtual Earth - http://www.microsoft.com/virtualearth/product/terms.html

If you have any questions would like to find out more about a MWS/VE Agreement, please contact maplic@microsoft.com for information.

Google Maps - http://code.google.com/apis/maps/terms.html

7. Miscellaneous: This License Agreement, the License granted hereunder, and the Software may not be assigned or in any way transferred without the prior written consent of Ektron. This Agreement and its performance and all claims arising from the relationship between the parties contemplated herein shall be governed by, construed and enforced in accordance with the laws of the State of New Hampshire without regard to conflict of laws principles thereof. The parties agree that any action brought in connection with this Agreement shall be maintained only in a court of competent subject matter jurisdiction located in the State of New Hampshire or in any court to which appeal therefrom may be taken. The parties hereby consent to the exclusive personal jurisdiction of such courts in the State of New Hampshire for all such purposes. The United Nations Convention on Contracts for the International Sale of Goods is specifically excluded from governing this License. If any provision of this License is to be held unenforceable, such holding will not affect the validity of the other provisions hereof. Failure of a party to enforce any provision of this Agreement shall not constitute or be construed as a waiver of such provision or of the right to enforce such provision. If you fail to comply with any term of this License, YOUR LICENSE IS AUTOMATICALLY TERMINATED. This License represents the entire understanding between the parties with respect to its subject matter.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, THAT YOU UNDERSTAND THIS AGREEMENT, AND UNDERSTAND THAT BY CONTINUING THE INSTALLATION OF THE SOFTWARE, BY LOADING OR RUNNING THE SOFTWARE, OR BY PLACING OR COPYING THE SOFTWARE ONTO YOUR COMPUTER HARD DRIVE, YOU AGREE TO BE BOUND BY THIS AGREEMENT'S TERMS AND CONDITIONS. YOU FURTHER AGREE THAT, EXCEPT FOR WRITTEN SEPARATE AGREEMENTS BETWEEN EKTRON AND YOU, THIS AGREEMENT IS A COMPLETE AND EXCLUSIVE STATEMENT OF THE RIGHTS AND LIABILITIES OF THE PARTIES.

Version 7.6.6, May 2009.

Guide Revision 2.0

Contents

Contents

Contents

1 Overview

An introduction to the benefits of PageBuilder

As the name suggests, PageBuilder is a tool that enables you to build Web pages in the Ektron CMS400.NET. What makes PageBuilder unique as a Web page creation tool is two-fold:

- **Firstly**, it allows non-technical users a simple way to build rich and fully-featured Web pages.
- **Secondly**, the Ektron development community can leverage PageBuilder as a simple way to reuse and share common functionality from one Ektron-powered site to another.

PageBuilder leverages the versatility of Ektron Portal Framework to streamline the process of building new pages on your Web site. Members of an organizations' Web team can now more efficiently make changes to pages and content, without relying on the availability of developers. Ultimately, PageBuilder helps not only redefine roles within an organization, but also maximizes efficiency and productivity.

In a typical PageBuilder scenario, the initial layout is managed by developers, while the final content, design, and placement of functionality is managed by non-technical users who have a need for creating specific Web pages. These non-technical, content authors would typically be marketing departments or similar organizations.

PageBuilder for Developers

To someone new to PageBuilder, it might seem as though PageBuilder takes the work of building and maintaining a Web site out of the IT department's hands. However, this is not the case. Web page developers still create page templates, building *wireframes* into which content authors place functional *widgets* that will make the Web page a success. PageBuilder allows the IT department to concentrate on what they do best: develop the back end of a system and address the technical nuances that today's Web sites generate. Content and messaging is out of their hands and squarely where it belongs: in Marketing.

Wireframes, Dropzones, and Widgets

Page Builder requires developers to create a *wireframe* template. This wireframe is the basic architecture for a Web page. As developers build wireframes, they add *dropzone* user controls where non-technical users need to insert the content, design, and messaging of the site. These zones are literally the areas into which someone can "drop" a *widget*.

Widgets are mini-applications that can provide either specific functionality (calculators, search, and social bars, etc) or areas into which you can add Ektron CMS400.NET content (content blocks, list summaries, collections, etc.). It is simple for developers to apply classes to these pages. Developers can also manage the level of control non-technical users have. For example, a developer can configure hard limits for the width of dropzones.

After a wireframe is created, an administrator assigns it to a folder, creates a page, and selects the widgets that will be available to place on that page.

PageBuilder for Everyday Users

Marketing teams (technical and non-technical alike) can build out entire pages on the wireframe templates by dragging-and-dropping widgets. This creates the user experience on the page while maintaining a consistent "look and feel." With PageBuilder, they can launch campaigns as needed and respond to market conditions rapidly with unique Web pages that have targeted and effective content.

PageBuilder pages have the same business-level controls as content blocks. Like other Ektron CMS400.NET content, PageBuilder pages maintain permissions, approval chains, enable SEO (through metadata), Taxonomy, Aliasing, and allow users to view histories and restore past versions.

Approvers can preview PageBuilder pages before they go live, and pages can be cloned and then modified to maintain look and feel across campaigns or to support A/B split testing.

Streamline Process and Improve Efficiency

PageBuilder adds a level of agility to key processes that overlap marketing and IT departments. By using it to streamline the workflow of launching new pages, time and expenses can be minimized. Moreover, your IT infrastructure remains secure because non-IT resources do not need to access mission-critical servers — all needed assets are accessible through the Workarea.

The CMS400.NET PageBuilder is the logical evolution of the WYSIWYG editor, extending the concept of content management beyond single elements of content. Now, that philosophy can be applied to whole pages and even entire sites, boosting productivity and giving you the tools you need to make your Web site do exactly what you want it to do.

2 Building Pages

Use case: AcmeBooks.com

AcmeBooks.com

AcmeBooks.com is a fast growing company that specializes in foreign and hard-to-find books. Recognizing it needed a stronger presence on the Web, it recently installed and deployed Ektron CMS400.NET.

The **Chief Information Officer** of Acme Books is especially excited about the company's recent move to CMS400.NET. She is particularly interested in PageBuilder, feeling it will enable the Marketing team to respond to changing market conditions and quickly launch and communicate promotions on the AcmeBooks.com Web site. She is also pleased that PageBuilder promises to appease the company's Web site developers. No longer will they have to deal with numerous daily requests for content changes from Marketing.

So, she wonders, "How does it all work? How will the Web team work together to use PageBuilder?" Right now, the team is composed of herself and:

Pete. He runs the Web **Development** team (including designers).

Grace. She is the **CMS/Web site administrator**.

Pierre. Acme Books' Marketing guru. He is also in charge Marketing and Acme Books' content authors.

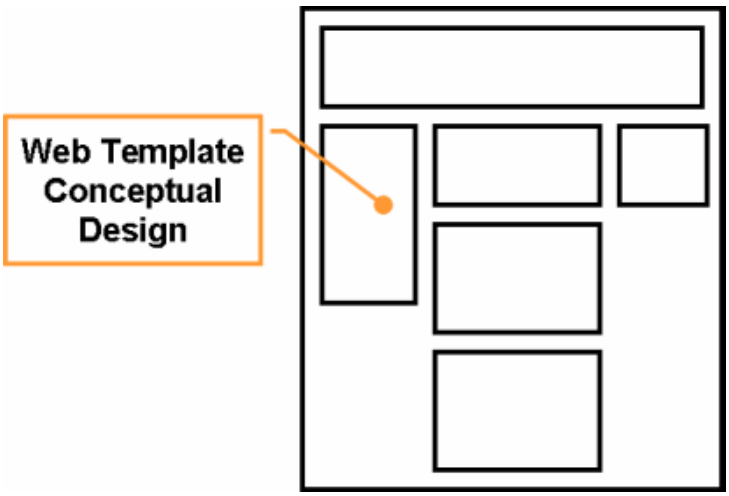
PageBuilder Workflow

Let's start with the general PageBuilder workflow. Basically, there are three stages:

Envisioning the "Big Picture"

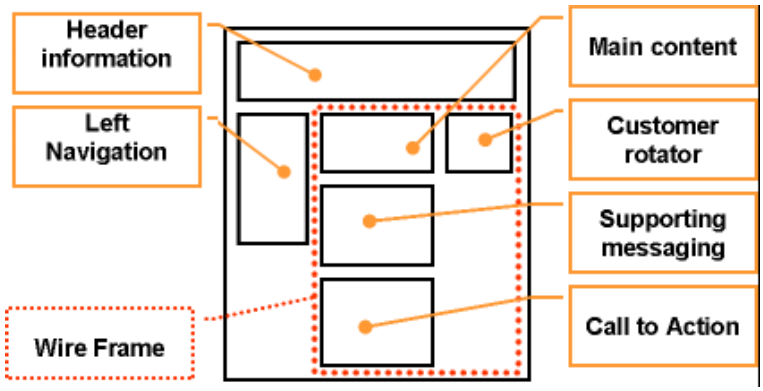
In the beginning, stakeholders define functionality and "look" needed on Web pages. At Acme Books, this includes the CIO, the Marketing team, the CMS400.NET Administrators, and the Web designers and developers in the IT department.

When a consensus is reached, Pete's designers and developers define the page layout and present it to the whole team for review and approval.



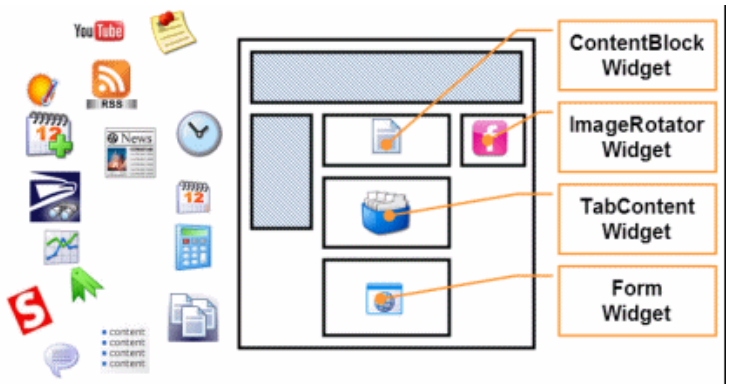
Designing and Development

Here, Pete's developers identify page areas and build wireframes based on design specifications. At this point, they should understand the functionality needed and use, modify or build widgets based on the requirements.



Implementation and Maintenance

When complete, a PageBuilder page is active and in use by Pierre and his team on a daily basis. Pierre can create pages, drag-and-drop widgets, and edit properties as necessary. Subject matter experts create and maintain content.



Steps to Creating a "PageBuilder" Page

Creating a Web page with PageBuilder functionality at Acme Books consists of the following tasks:

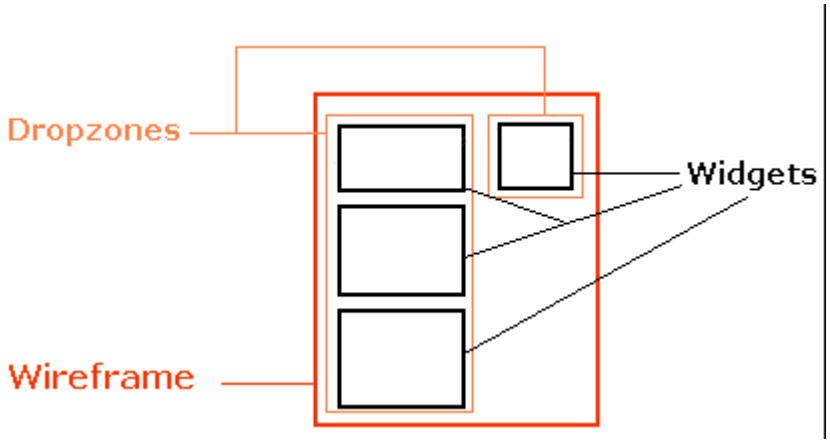
- "Create a PageBuilder Wireframe" on page 5. This is performed by Pete and his developers.
- "Enable Manual Aliasing" on page 10. This is performed by Grace, the Web site Administrator.
- "Identify the PageBuilder Wireframe in the CMS" on page 11. This is performed by Grace, the Web site Administrator.
- "Assign the PageBuilder Wireframe to a Folder" on page 12. This is performed by Grace, the Web site Administrator.
- "Create the New Page" on page 13. This is performed by Pierre and his team.

- "Place Widgets on the Page" on page 16. This is performed by Pierre and his team.

Create a PageBuilder Wireframe

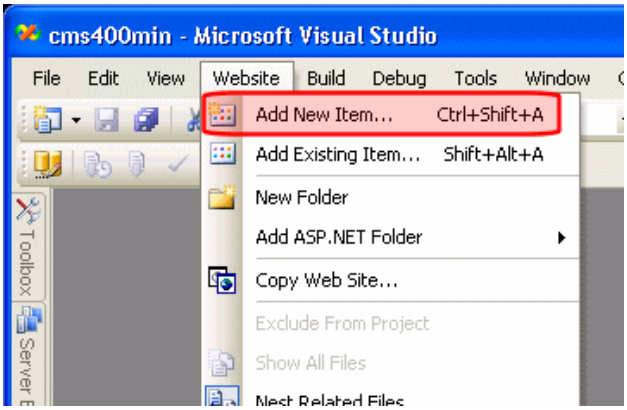
As explained earlier, Pete first creates a *wireframe*. Next, he defines *dropzones*, areas of the page on which a content creator drags and drops *widgets*.

The relationship between a wireframe, a dropzone, and a widget is illustrated below.

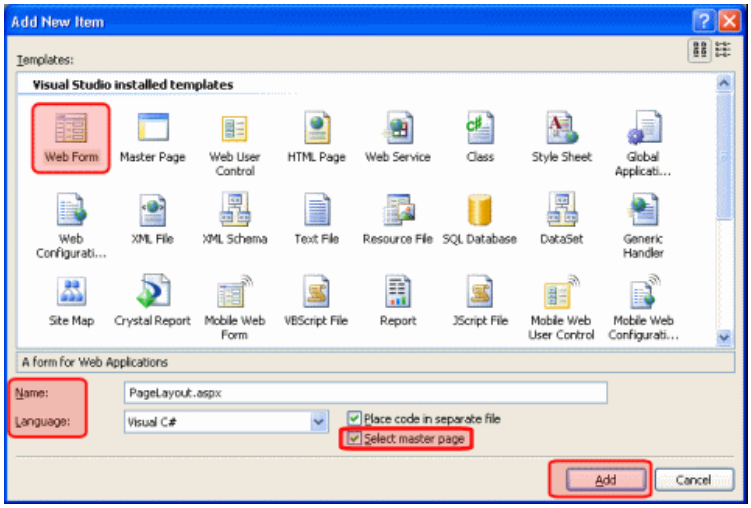


Here is how Pete creates a wireframe that contains one dropzone.

1. He opens the Web site in Visual Studio.
2. He adds a new Web form to the site by clicking **Website > Add New Item**.



3. He selects **Web Form**.
4. Pete sets the Name to PageLayout.aspx, the Language to Visual C#, and checks Select master page.



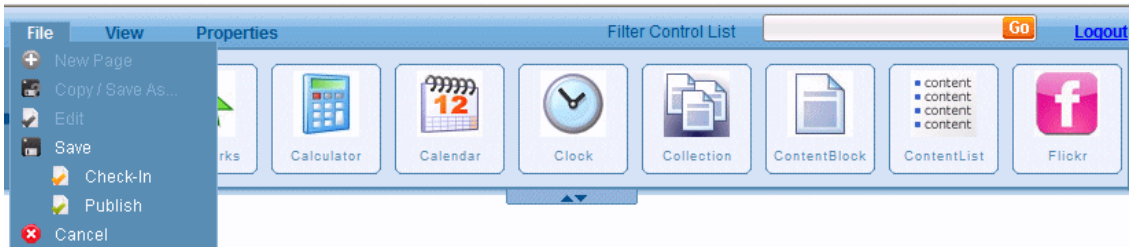
5. He clicks **Add**, then changes to **Source view**.

6. He registers the following assemblies directly below the @ Page directive (at the top of the file).

```
<%@ Register Assembly="Ektron.Cms.Widget" Namespace="Ektron.Cms.PageBuilder" TagPrefix="PB" %>

<%@ Register Assembly="Ektron.Cms.Controls" Namespace="Ektron.Cms.Controls" TagPrefix="CMS" %>
```

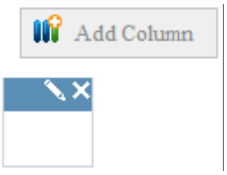
7. Next, he'll drop a PageBuilder Menu and DropZone user control onto the page. When rendered, a PageBuilder menu and control look like this.

.



This control lets a content author drop widgets on the page. It also provides the save/check in/publish functions, and lets the author preview how the page will look when it's published.

When rendered on a page, a DropZone user control looks like this.



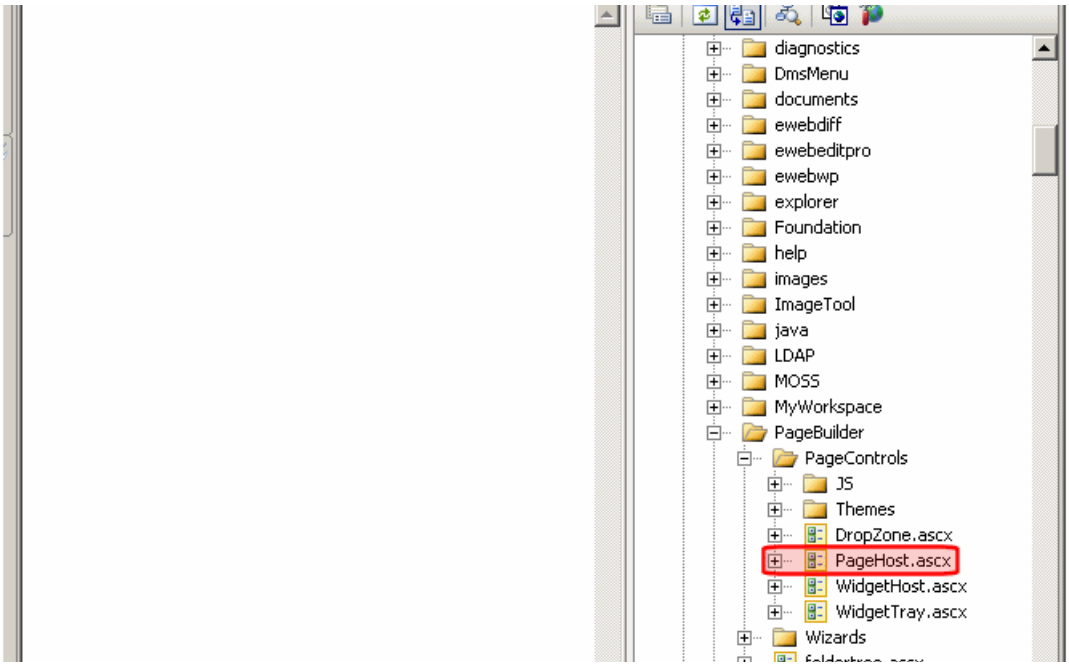
The content author uses a DropZone control as a placeholder, into which he will insert widgets. He can also use it to insert additional placeholders as

needed.

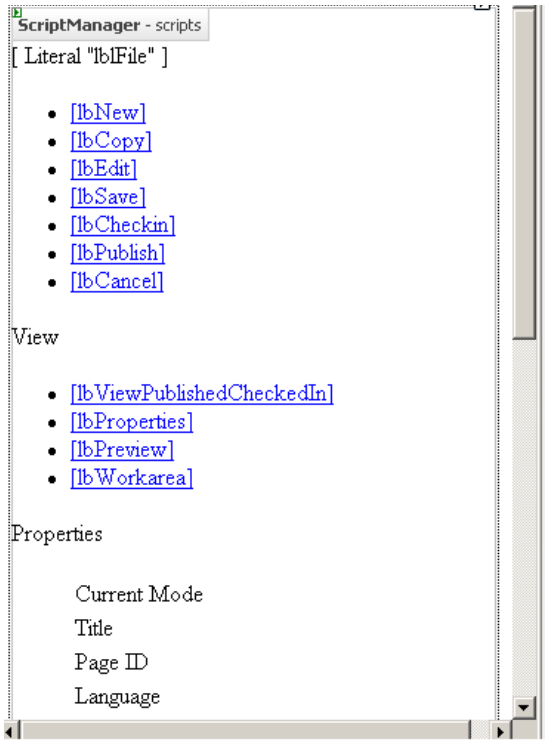
8. He switches Visual Studio from Source to Design view.

9. In the Visual Studio explorer, he finds the `/Workarea/PageBuilder/PageControls` folder.

10. He drags and drops the PageHost.ascx user control onto the page.



11. The left panel now looks like this.



12. Below the PageHost control (below Pullchain), he drags and drops the Dropzone.ascx user control onto the page.
13. He switches to source view.
14. The screen now looks like this.

Note: See "Advanced Page-Builder Topics" on page 59 for information on customizing user controls.

```
<%@ Register Assembly="Ektron.Cms.Widget" Namespace="Ektron.Cms.PageBuilder" TagPrefix="PB" %>
<%@ Register Assembly="Ektron.Cms.Controls" Namespace="Ektron.Cms.Controls" TagPrefix="CMS" %>

<%@ Register Src="Workarea/PageBuilder/PageControls/PageHost.ascx" TagName="PageHost"
    TagPrefix="uc1" %>
<%@ Register Src="Workarea/PageBuilder/PageControls/DropZone.ascx" TagName="DropZone"
    TagPrefix="uc2" %>


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <uc1:PageHost id="PageHost1" runat="server">
        </uc1:PageHost>
        <uc2:DropZone id="DropZone1" runat="server">
        </uc2:DropZone></div>
    </form>
</body>
</html>
```

Note that Pete adds only one dropzone now. He plans to add more later.

15. Pete is finished with the user control file and is ready to begin working on his codebehind file.

16. Pete opens the codebehind page, PageLayout.aspx.cs.

17. He adds a reference to the PageBuilder namespace by adding the following line after the last using statement.

```
using Ektron.Cms.PageBuilder;
```

18. Inherit the PageBuilder class instead of System.Web.UI.Page. To do this, change:

```
public partial class PageLayout : System.Web.UI.Page
```

To:

```
public partial class PageLayout : PageBuilder
```

19. Pete adds the following code, which handles errors and notifications, after the Page_Load event.

Tip! You can copy the following code from **siteroot/developer/pagebuilder/pagelayout.aspx.cs.**

```
public override void Error(string message)

{

jsAlert(message);

}
```

Note: Ektron Best Practice is to set values for the `FolderID` and `SelTaxonomyID` properties. See "How a Page's Default Folder is Set" on page 59 and "Assigning a Default Taxonomy to a Wireframe" on page 63

```
public override void Notify(string message)

{

jsAlert(message);

}


public void jsAlert(string message)

{

Literal lit = new Literal();

lit.Text = "<script type=\"\" language=\"\">{0}</script>";

lit.Text = string.Format(lit.Text, "alert('" + message + "');");

Form.Controls.Add(lit);

}
```

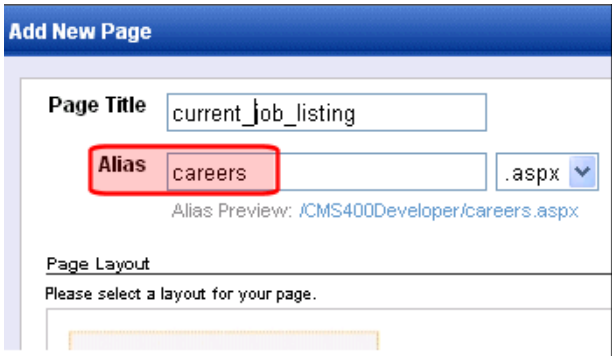
He does not need to use the jsAlert system defined here. If he doesn't, Pete must somehow add overrides to handle errors and notifications to the codebehind.

20. Save the PageLayout.aspx and PageLayout.aspx.cs files.

Enable Manual Aliasing

Note: The Ektron "best practice" is to enable Manual and Auto aliasing.

Pete's development team needs to make sure that manual aliasing is enabled within Ektron CMS400.NET. Doing this allows you to apply a user-friendly URL when you're creating a PageBuilder page, as shown below.

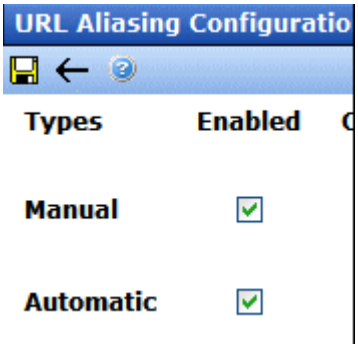


Note: See "URL Aliasing" in the *Administrator Guide* for more information.

To enable manual aliasing, the development team does these steps.

1. In the Ektron CMS400.NET Workarea, they go to **Settings > Configuration > URL Aliasing > Settings**.

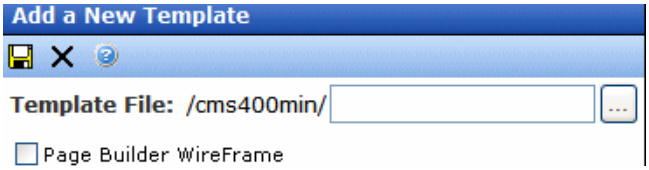
2. They select Manual, then click Save ( ).



Identify the PageBuilder Wireframe in the CMS

In this step Grace, the CMS administrator, makes the wireframe that Pete created available within the Ektron CMS400.NET Workarea.

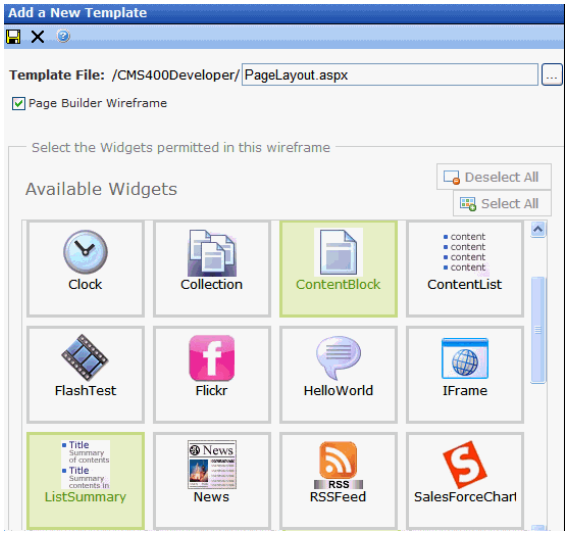
1. In the Ektron CMS400.NET Workarea, she goes to **Settings > Configuration > Template Configuration**.
2. She clicks the Add button ( ).
3. The Add a New Template screen appears.



4. Grace clicks the browse button [...] and navigates to PageLayout.aspx, the wireframe Pete created earlier in "Create a PageBuilder Wireframe" on page 5.
5. She clicks the PageBuilder Wireframe check box. This box tells Ektron CMS400.NET that this template can be used to create a PageBuilder page.

   All widgets that can be applied to the template appear. Ektron CMS400.NET provides over 30 standard widgets. See "Standard Widgets" on page 49.

Note: "Creating your own Widgets" on page 29 explains how to create a custom widget.



6. Grace clicks the ContentBlock and List Summary widgets. When she does, their background color changes.

7. She clicks **Save** ( ).

Assign the PageBuilder Wireframe to a Folder

Here, Grace creates a folder for the content that will appear on the new page. To accomplish this, she does the following.

1. Grace goes to **Content** in the Ektron CMS400.NET Workarea. All folders appear in the left panel.

2. She clicks the Root folder (highlighted below).

Note: For more information about folder permissions and approval chains, see "Setting Permissions" in the *Administrator Guide*.

3. She clicks **New > Folder**.

   Grace knows that it is an Ektron best practice to keep content and PageBuilder pages in separate folders, so she will later create and configure a folder called "Content". The content folder will simply contain content blocks and other assets, as well as sub-folders.

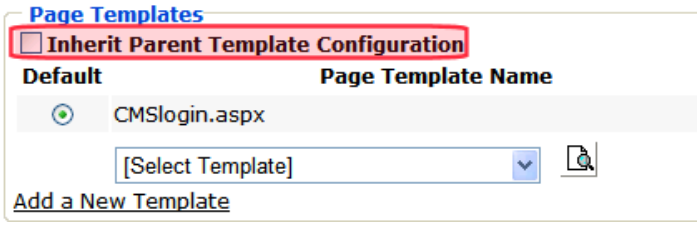
4. She accesses the folder properties for the new folder.

5. In the Foldername field, she types Pages.

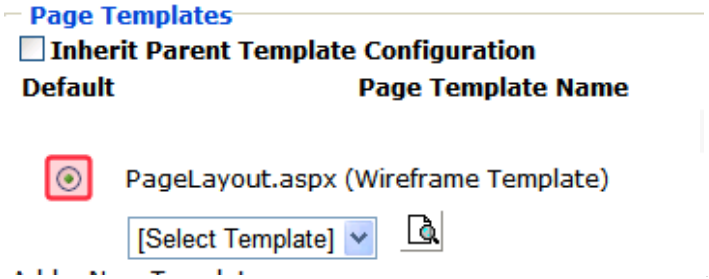
Next, Grace will assign the template created earlier as the default (and only) template for this folder. This will ensure that only PageBuilder pages can be created in this folder.

6. In the Page Templates area (circled below), Grace clears the **Inherit Parent Template Configuration** option and clicks OK at the prompt.

Note: For more information about folder inheritance, see "Setting Permissions" in the *Administrator Guide*.



7. She selects PageLayout.aspx from the template list. This is the wireframe Pete created earlier.
8. She clicks **Add** (to the right of the pull-down menu). PageLayout.aspx is now added to the list of page templates.
9. She deletes **CMSlogin.aspx** from the list to ensure that no content is placed in the folder.
10. Grace selects **PageLayout.aspx** (the default option, circled below) and clicks **Save**( ).



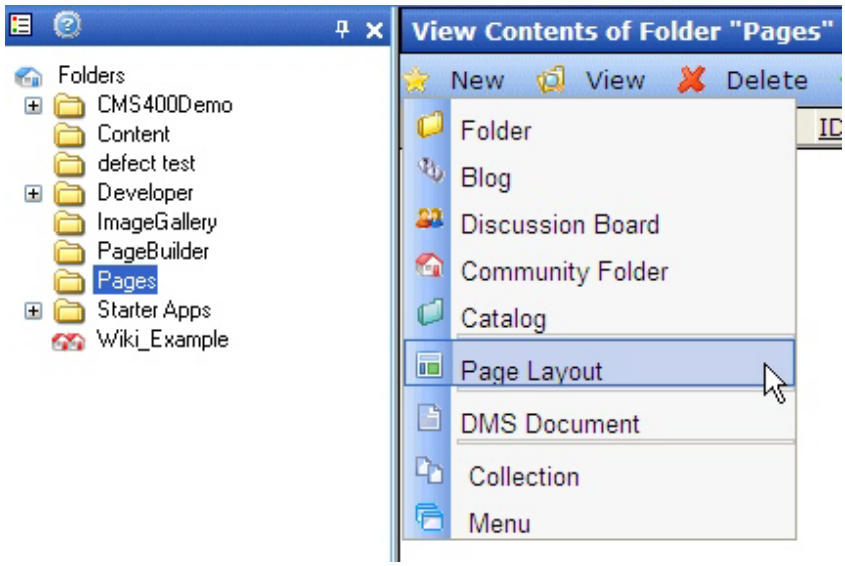
11. Grace will follow steps 3 through 6 to create a folder for content. However, this time, instead of assigning pagelayout.aspx as the template, she will assign a template used to create Ektron CMS400.NET content.

Create the New Page

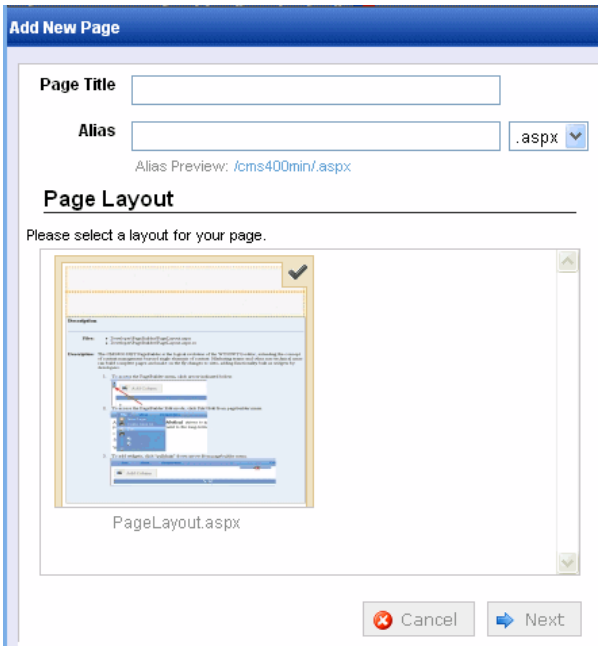
It's time for Pierre and his team to create a PageBuilder page. Currently, at AcmeBooks.com, there is a pressing need to market a new series of mystery novels from an up-and-coming Australian novelist. So, the Marketing team needs to quickly create a Web page to announce the new series. Here's how they build the page with PageBuilder.

Important: When you edit Pagebuilder pages in the Ektron CMS400.NET Workrea via Internet Explorer, IE version 7 or higher is required.

1. They click the Pages folder, which was created in "Assign the PageBuilder Wireframe to a Folder" on page 12.
2. They click **New > Page Layout**.



3. The Add New Page screen appears.



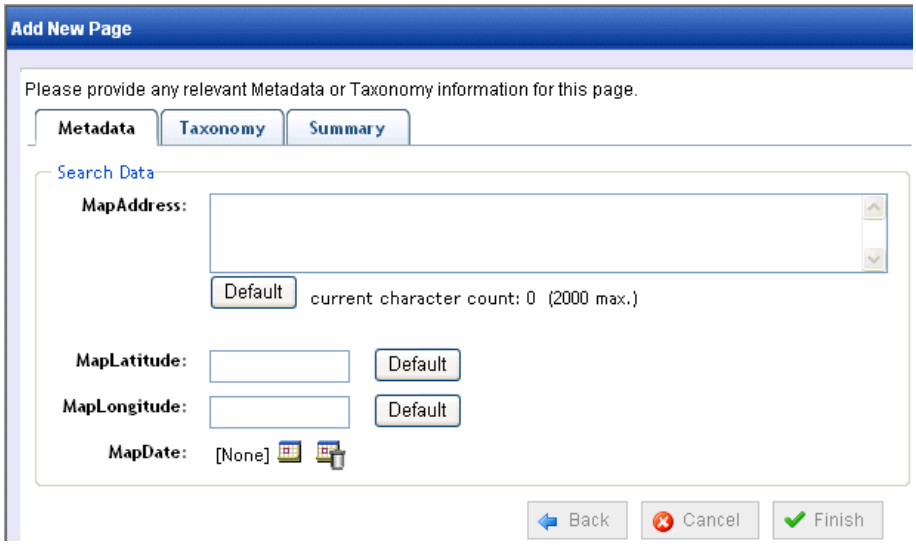
4. The team enters **New Australian Mystery Series** in the Page Title field.
5. By default, the Alias matches the Page Title. The Marketing team changes it to **Australian_Mystery**.

They do this in the Alias field so the new page has a user-friendly name. For example, by default, the page's name is `www.acmebooks.com/new_mystery_series.aspx`. The alias Australian_Mystery lets the page display as `www.acmebooks.com/Australian_Mystery`.

6. In the Page Layout area, the Marketing team clicks PageLayout.aspx, the wireframe assigned to this folder in "Assign the PageBuilder Wireframe to a Folder" on page 12. A green checkmark and background color indicate this wireframe will be used to create the page.

Note: All wireframe templates you assign to a folder's properties appear in the Page Layout area.

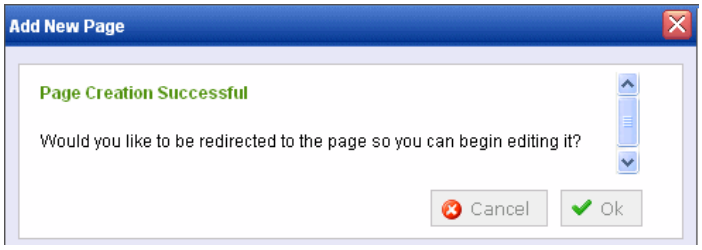
7. After clicking Next, the following screen appears.



This allows the team to assign metadata, taxonomy categories, and a summary to the page.

8. Click **Finish**.

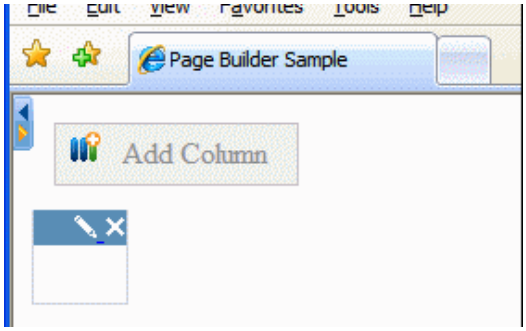
9. The following message appears. Press OK.



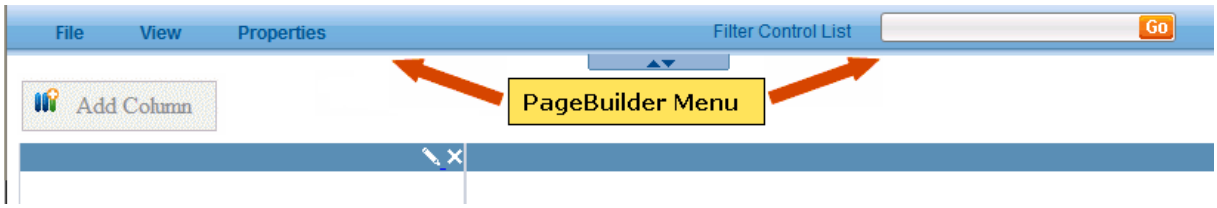
Place Widgets on the Page

Note: For information about how to create a widget, see "Creating your own Widgets" on page 29.

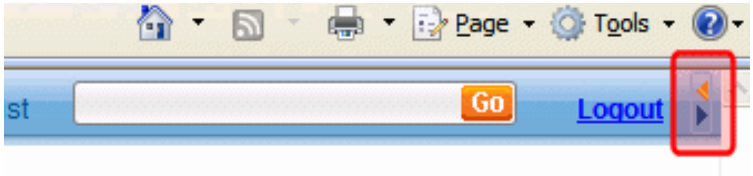
After Pierre and the Marketing team create the page, a new page opens. It contains one dropzone, which consists of one Add Column button and one column (shown below).



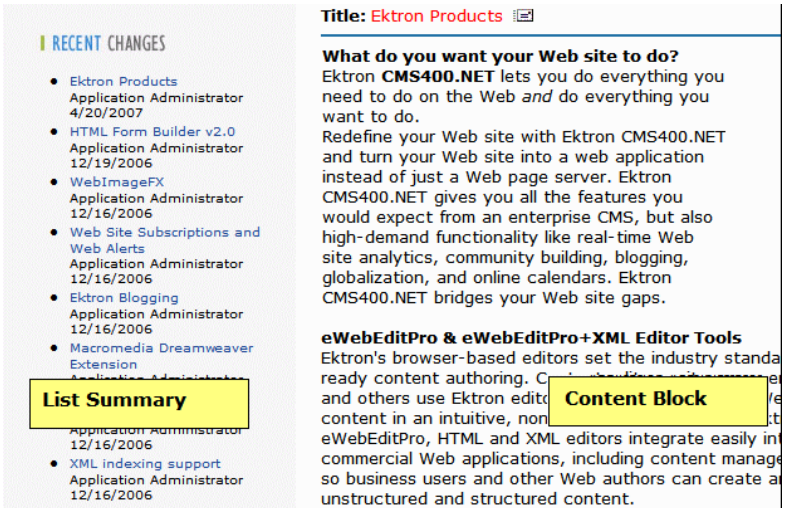
The top of the page contains a PageBuilder menu control. Pierre will use it to perform actions on the page, such as check in, view properties, and drop widgets onto the page.



Pierre uses the right/left arrow buttons (circled below) to open and close the PageBuilder menu.



Pierre determines that the page needs a two-column layout, as shown below.



Note: Although you can drop existing content onto the page, the Marketing team will create new content on the page.

- The left column is 35% wide and displays a list of all content in the Marketing folder. Web site visitors use the list to access all Marketing collateral for AcmeBooks.com.
- The right column is 65% wide and displays a single content block.

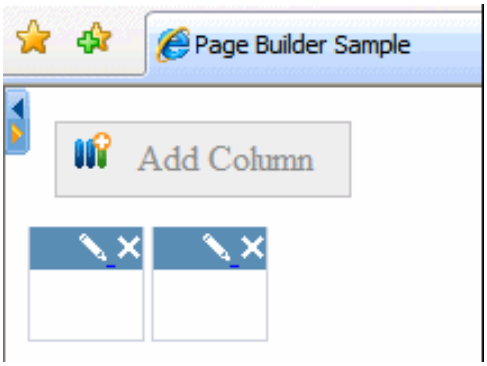
Pierre's Marketing team creates this layout in three steps:

- "Part 1: Add New Column and Set Column Widths" on page 17
- "Part 2: Insert New Content Block Widget into Right Column" on page 18
- "Part 3: Insert a List Summary Widget into Left Column" on page 22

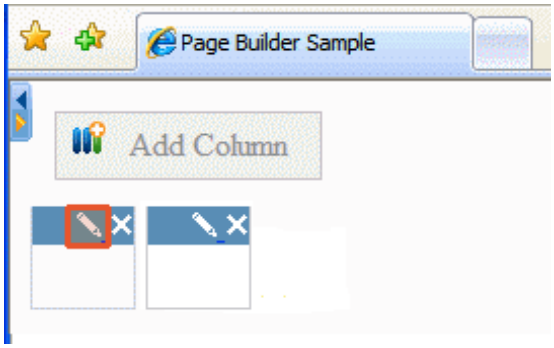
Part 1: Add New Column and Set Column Widths

In this part of the procedure, the Marketing team adds a second column then sets the width of both columns.

1. They click the Add Column button, and a new column appears to the right of the existing one.



2. They set the left column width to 35% by clicking the pencil icon (circled below).

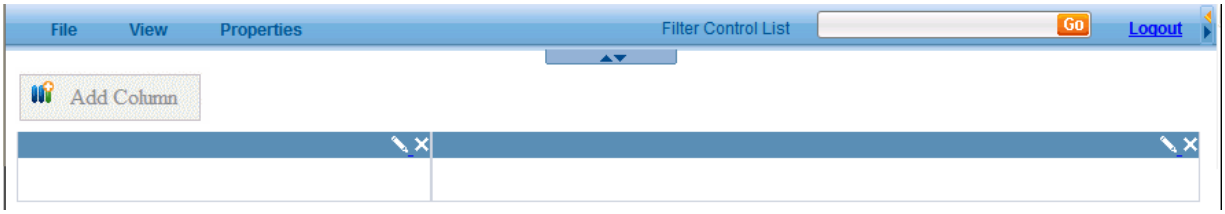


3. A field lets them enter a width and measurement (pixels or percent).

4. So, they enter 35 into the New Width field, change the measurement to Percent, and click Save.

5. Then, the team sets the right column to 65% width by clicking the pencil icon, changing its width to 65 Percent, and clicking Save.

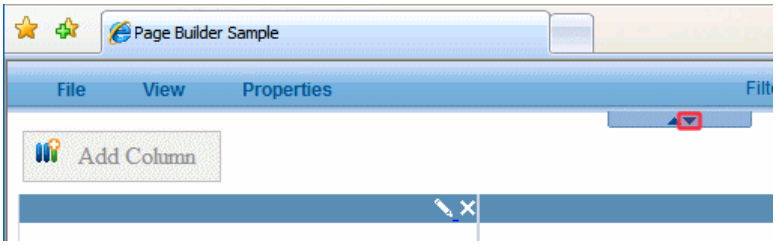
6. Now the screen looks like this.



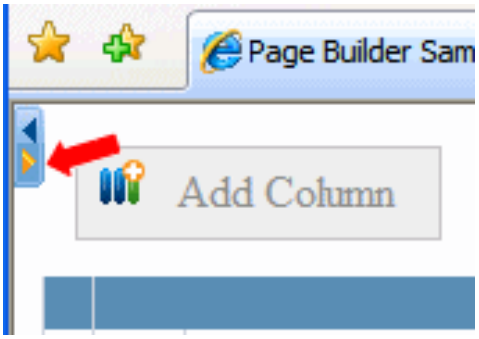
Part 2: Insert New Content Block Widget into Right Column

Now that Pierre's team has set up the columns, Pierre is eager to try inserting a ContentBlock widget into the right column.

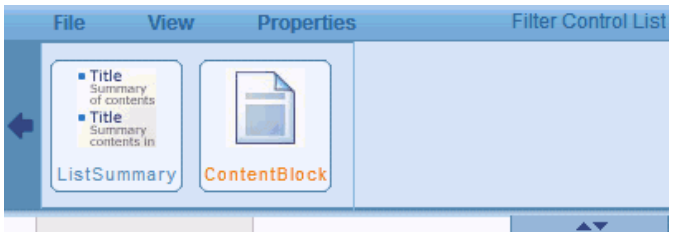
1. From the small block in the center of the PageBuilder menu, Pierre clicks the down arrow (circled below).



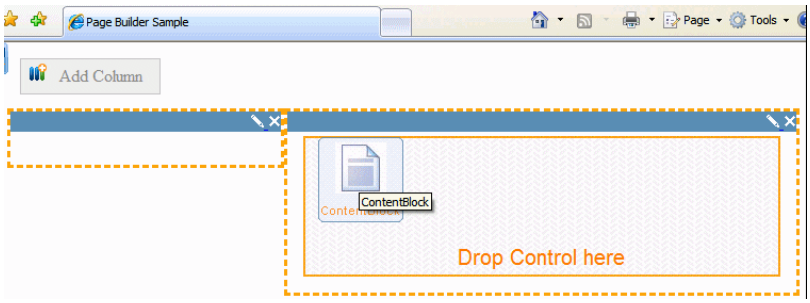
Note: Pierre tells his team that if they do not see the PageBuilder Menu, look for small block with two arrows, one orange and one blue (shown right). Click the orange arrow to open the menu.



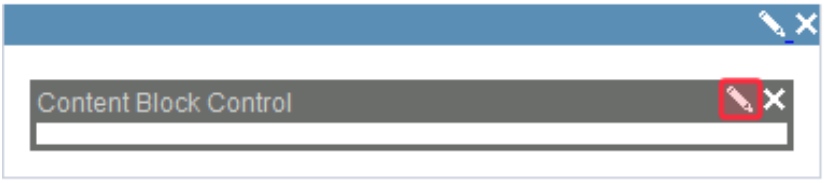
2. All widgets that the Web site administrator (Grace) assigned to the page wireframe in "Identify the PageBuilder Wireframe in the CMS" on page 11 appear.



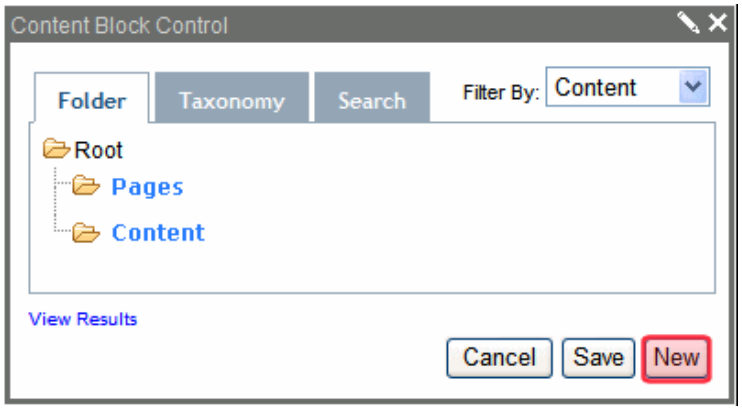
3. Pierre places the cursor over the content block widget and drags it below the widget panel.
4. The widget panel disappears.
5. Pierre drops the content block widget in the right column.



6. That column looks like this.

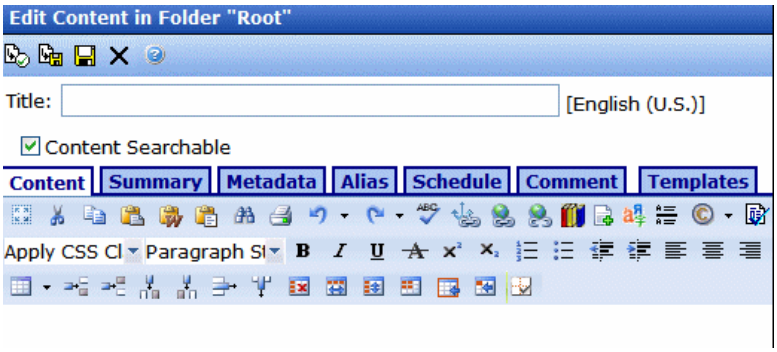


7. Pierre clicks the pencil icon (circled above), and the following screen appears.



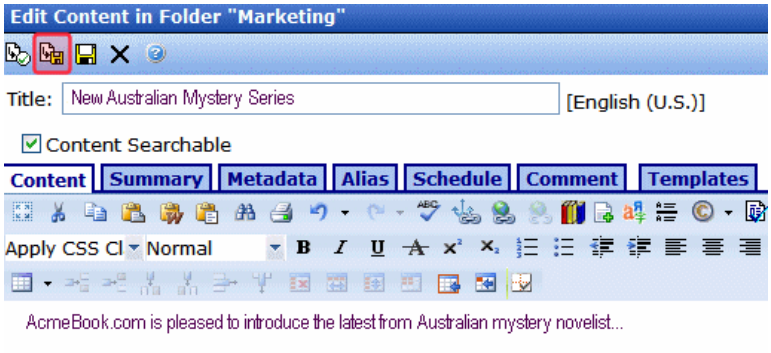
8. He selects the Content folder, and clicks New in the lower right corner (circled above).

9. The Add Content screen appears.

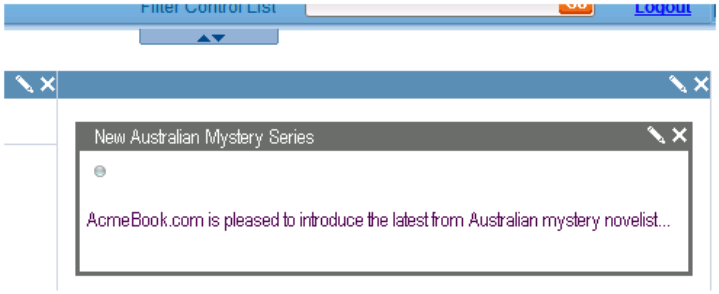


10. In the Title field, Pierre enters “New Australian Mystery Series”.

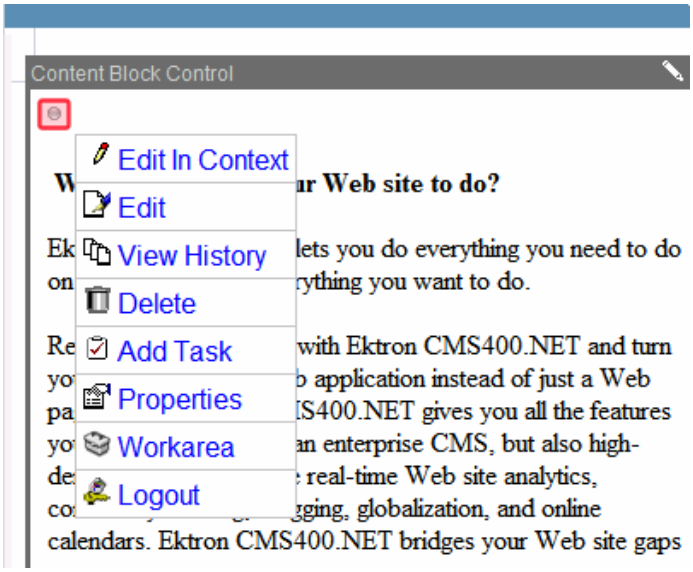
11. In the content area, he enters some text about the series of books, and clicks Check In.



12. The content displays as shown below.



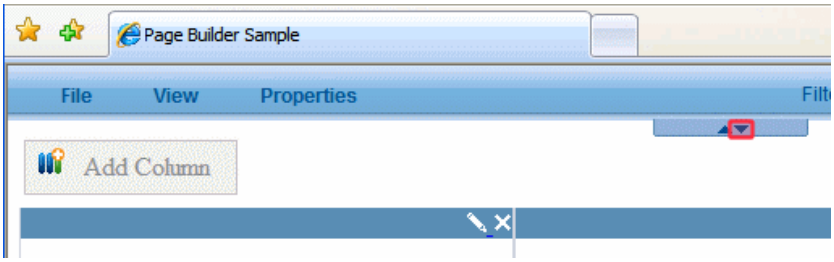
13. When the Marketing team wants to edit content on this page, they can click the Access Point in the upper left corner. A menu displays.



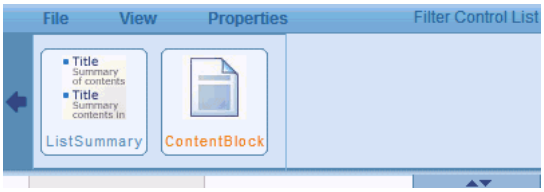
Part 3: Insert a List Summary Widget into Left Column

Now that Pierre has inserted content in the right column, he can insert a List Summary widget in the left column.

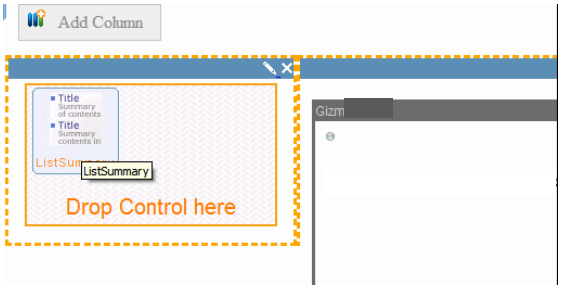
1. From the PageBuilder menu, he clicks the down arrow (circled below).



2. All widgets Grace (the CMS administrator) assigned to the page wireframe in "Identify the PageBuilder Wireframe in the CMS" on page 11 appear.



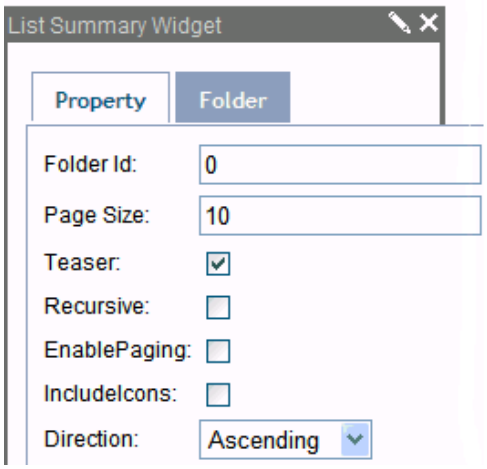
3. Pierre places the cursor over the List Summary widget and drags it below the widget panel.
4. The widget panel disappears.
5. He drops the List Summary icon in the left column.



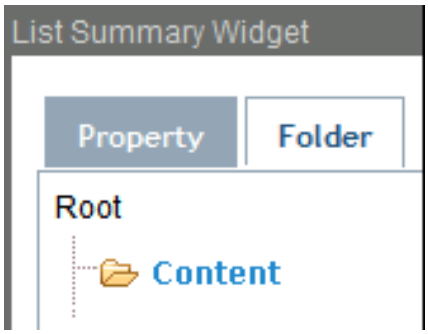
6. That column looks like this.

7. The widget displays a list summary for content in the root folder. Pierre needs to change it to refer to the Content folder. To do that, he clicks the pencil icon in the widget's top right corner (circled above).

8. The following screen appears.

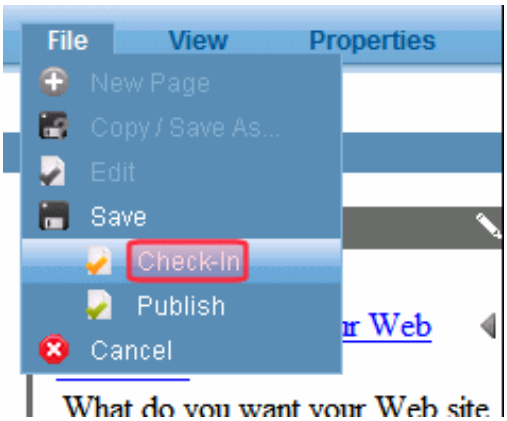


9. He clicks the Folder tab and selects the Content folder.



10. He clicks Save, and the list summary shows content in the Content folder.

11. From the PageBuilder menu, Pierre clicks **File > Check In**.



12. The whole Marketing team can now review the page in Preview mode and make sure it is formatted properly.

13. If they need to change it, they can click **File > Edit** from the PageBuilder menu.

Final Assessment

The CIO of AcmeBooks.com is can see the power of the PageBuilder, and is convinced that this will greatly increase effectiveness of the company's Marketing and sales goals. The simple workflow and ease of use will help the Marketing team quickly deploy and launch marketing campaigns and keep the Web site fresh and dynamic.

She still has one lingering question:

"How on earth do you build a widget?"

3 PageBuilder Code Samples

Working with PageBuilder code

The following shows a complete example of pagelayout.aspx

PageLayout.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="PageLayout.aspx.cs"
Inherits="Developer_PageBuilder_PageLayout" %>

<%@ Register Src="~/Workarea/PageBuilder/PageControls/PageHost.ascx" TagName="PageHost"

    TagPrefix="ucPageBuilder" %>

<%@ Register Src="~/Workarea/PageBuilder/PageControls/DropZone.ascx" TagName="DropZone"

    TagPrefix="ucPageBuilder" %>

<%@ Register Assembly="Ektron.Cms.Widget" Namespace="Ektron.Cms.PageBuilder"

TagPrefix="PB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

    <title>Page Builder Sample</title>

</head>

<body>

    <form id="form1" runat="server">

        <div>

            <ucPageBuilder:PageHost ID="ucCms400Developer" FolderID="382" runat="server" />

<PB:DropZone ID="top" AllowAddColumn="true" AllowColumnResize="true" runat="server">

<ColumnDefinitions>

<PB:ColumnData columnID="0" unit="percent" width="100" />

</ColumnDefinitions>

</PB:DropZone>

        </div>

        <div>
```

```
<PB:DropZone ID="bottom" AllowAddColumn="true" AllowColumnResize="true" runat="server">
<ColumnDefinitions>
<PB:ColumnData columnID="0" unit="percent" width="100" />
</ColumnDefinitions>
</PB:DropZone>
        </div>
        <div>
        </div>
    </form>
</body>
</html>
```

PageLayout.aspx.cs

The following shows a complete example of pagelayout.aspx.cs

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using Ektron.Cms.PageBuilder;
public partial class Developer_PageBuilder_PageLayout : PageBuilder
{
    protected void Page_Load(object sender, EventArgs e)
    {
```

```
    }
    public override void Error(string message)
    {
        jsAlert(message);
    }
    public override void Notify(string message)
    {
        jsAlert(message);
    }
    public void jsAlert(string message)
    {
        Literal lit = new Literal();
        lit.Text = "<script type=\"\" language=\"\">{0}</script>";
        lit.Text = string.Format(lit.Text, "alert('" + message + "');");
        Form.Controls.Add(lit);
    }
}
```

4 Creating your own Widgets

Using and creating widgets

What's a Widget?

Widgets are .NET user controls that allow a site visitor to perform a function on a Web page. They are small portions of code that developers can write once and then be reused multiple times by content authors across a site.

For example, a widget can provide a simple calculator or a stock ticker. Or, it can display significant information from Ektron CMS400.NET, such as a content block or a List Summary.

Widgets at AcmeBooks.com

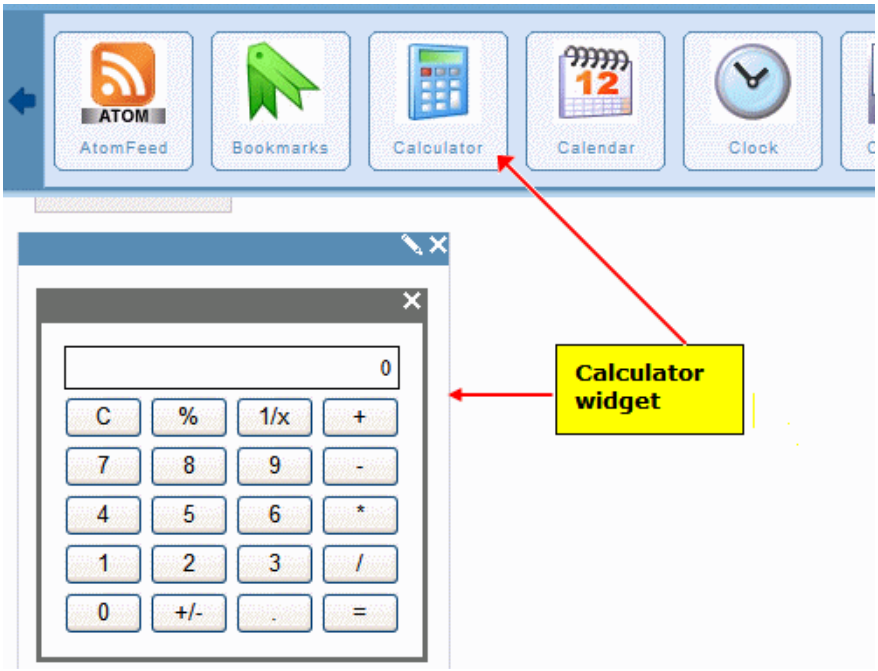
While this definition adds some clarity to what widgets are, the CIO and the Web team still have a lot of questions about how to use widgets, and how they are created.

So, let's get started with using and building widgets.

Using Widgets on a PageBuilder Page

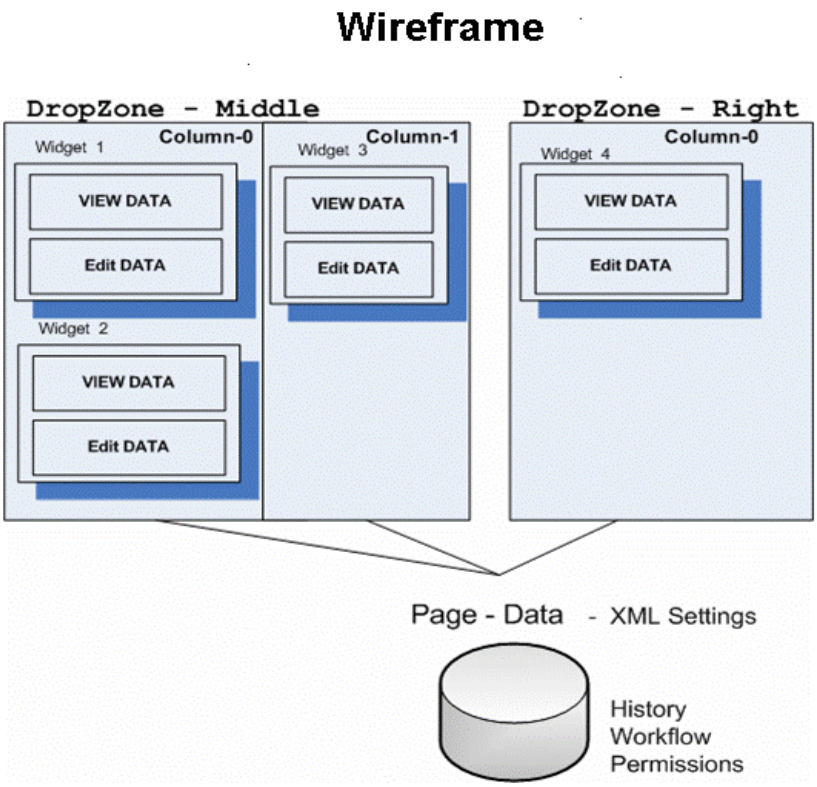
After widgets are integrated into CMS400.NET, users (like the Marketing team) can add them to a Dashboard in their profile page or a Community Group's page. They can also drag-and-drop these building blocks onto a PageBuilder page (as shown below).

Note: Over 30 standard widgets included in Ektron CMS400.NET. See "Standard Widgets" on page 49



Widgets, DropZones, and Pages

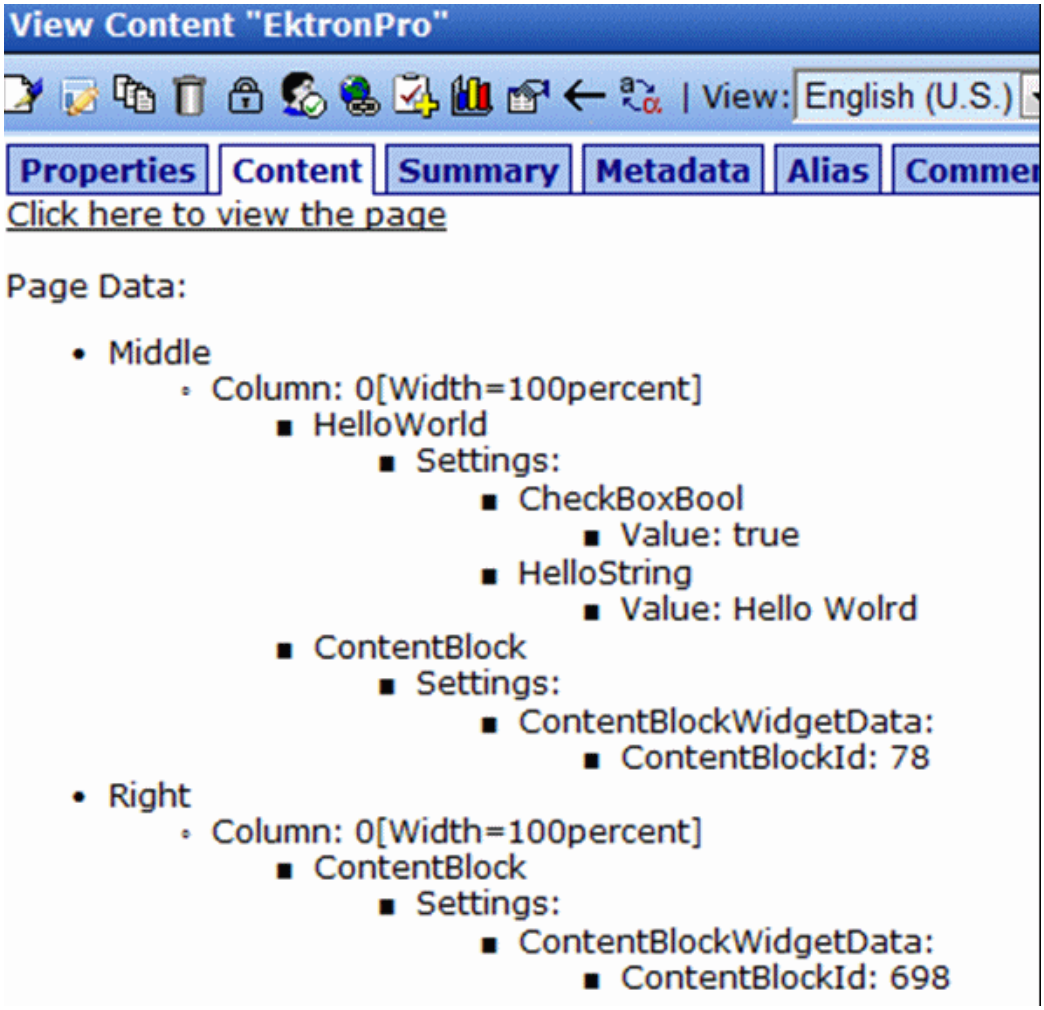
The relationship between a wireframe, dropzones, and widgets is shown below.



The above illustration of a wireframe depicts how:

- A wireframe can have several dropzones
- Each dropzone can have several columns
- Each column can have several widgets

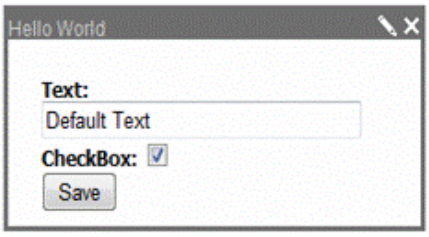
Ektron CMS400.NET stores each page's data (a serialized XML string) as a type of content within the Ektron CMS400.NET Workarea. The string is stored like other content types, such as HTML content and XML Smart Forms. The following illustration shows a widget's page data within the Workarea.



Widget States

Widgets placed on a PageBuilder page have three possible combinations of states.

Page mode	Widget mode	State	Illustration
View	View	Widget content appears on page	**Text:** Default Text **CheckBox:** True

Page mode	Widget mode	State	Illustration
Edit	View	Widget can be dragged/dropped, moved, deleted	
Edit	Edit	User defines widget information	

In a widget's user control file, you create an `asp:MultiView` element that determines available actions when a widget is in View mode and Edit mode. See sample below:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="HelloWorld.ascx.cs"
Inherits="widgets_HelloWorld" %>

<%@ Register Assembly="System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"

Namespace="System.Web.UI" TagPrefix="asp" %>

<asp:MultiView ID="ViewSet" runat="server" ActiveViewIndex="0">

    <asp:View ID="View" runat="server">

    <!-- You Need To Do ............................. -->

    <asp:Label ID="HelloTextLabel" runat="server"></asp:Label><br />

    <asp:Label ID="CheckBoxLabel" runat="server"></asp:Label>

    <!-- End To Do ............................. -->

</asp:View>

<asp:View ID="Edit" runat="server">

    <div id="<%=ClientID%>_edit">

    <!-- You Need To Do ............................. -->

    <asp:TextBox ID="HelloTextBox" runat="server" Style="width: 95%"> </asp:TextBox><br />

    <asp:CheckBox ID="MyCheckBox" runat="server" Checked="false" /> <br /><br />
```

```
    <!-- End To Do ............................. -->

    <asp:Button ID="CancelButton" runat="server" Text="Cancel" OnClick="CancelButton_Click" />
      

    <asp:Button ID="SaveButton" runat="server" Text="Save" OnClick="SaveButton_Click" />

    </div>

</asp:View>

</asp:MultiView>
```

Creating a Widget

A widget consists of three file types.

- .ascx - contains a widget's source code
- .ascx.cs or .vb - contains widget's codebehind
- .ascx.jpg - image that represents a widget in the widget selection tool

Note: Your widget might use additional files, such as .css or .js files. Ektron recommends placing these files in a folder within `site root/widgets`, and giving the folder the same name as the custom widget.

When creating a widget, you save files to the `site root/widgets` folder. This folder path is defined in the site root/web.config file, so if Pete needs to change the folder name or path, he must update the following web.config `ek_widgetPath` element:

```
<add key=”ek_widgetPath” value=”Widgets/" /> .
```

The “Hello World” Widget

To get a basic understanding of how to create a widget, Pete will create a simple widget, which will reside in the `site root/widgets` folder. This widget is based on the Hello World widget that is installed with the Developer Sample site. It uses the following files:

- HelloWorld.ascx - user control file
- HelloWorld.ascx.cs - user control codebehind file
- HelloWorld.ascx.jpg - image that represents this control on the widget menu

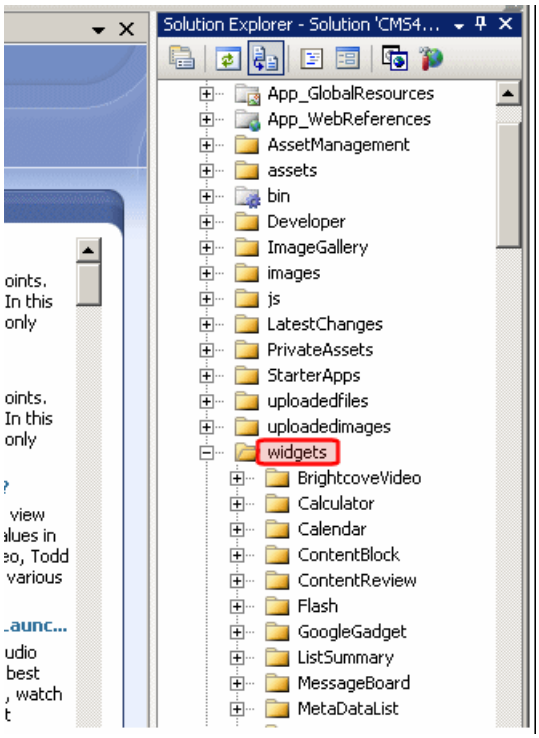
Pete will complete these tasks to create a widget.

- "Copy, Paste and Rename HelloWorld.ascx and HelloWorld.ascx.jpg" on page 33
- "Update the Class Names in the New Files" on page 37
- "Add Widget in Ektron CMS400.NET Workarea" on page 38

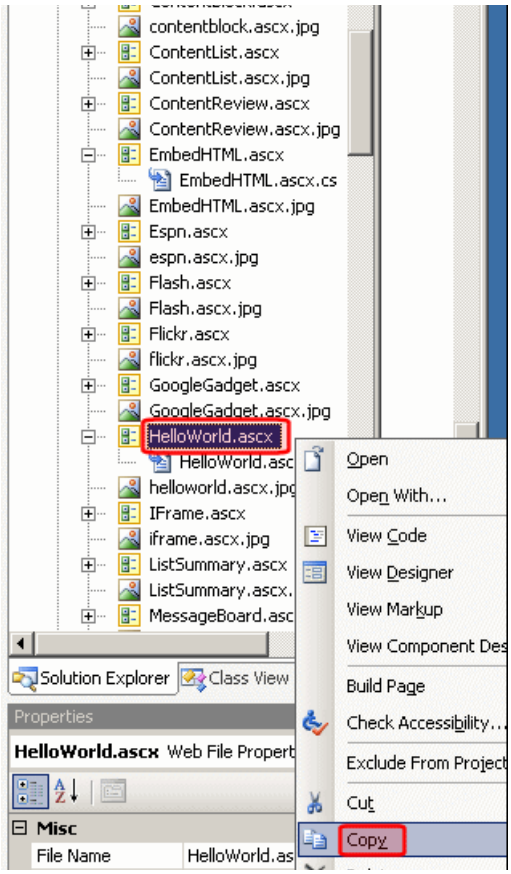
Copy, Paste and Rename HelloWorld.ascx and HelloWorld.ascx.jpg

1. Open your Web site in Visual Studio.

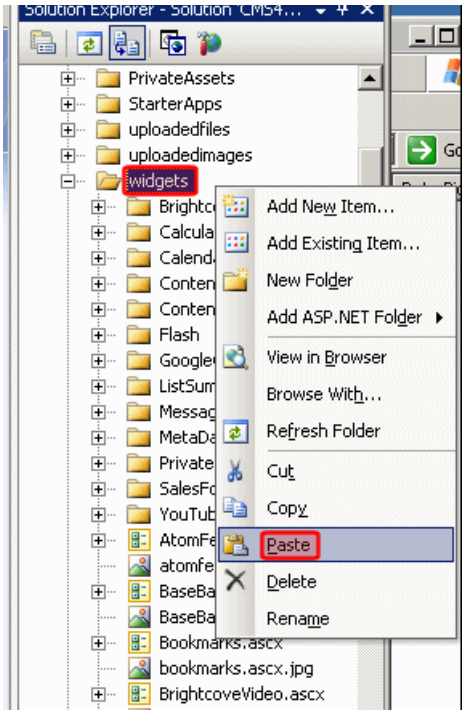
2. In the Visual Studio Solution Explorer, open the `widgets` folder.



3. Within that folder, scroll down to and select HelloWorld.ascx.

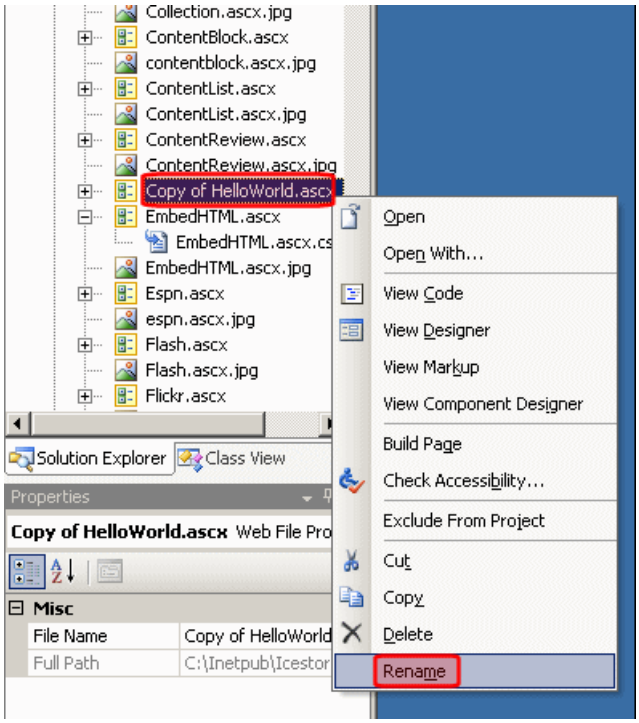


4. Right click the mouse and select Copy.
5. Scroll up to the `widgets` folder.
6. Right click the mouse and select Paste.



7. Scroll down until you see `Copy of HelloWorld.ascx`.

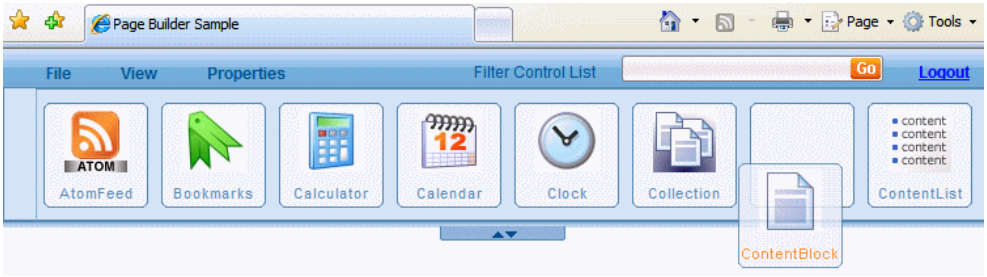
8. Right click the mouse and select Rename.



9. Rename the file `new_widget.ascx`. Notice that Visual Studio also renames the codebehind file to `new_widget.ascx.cs`.

Note: The image file is 48 x 48 pixels and 72 dpi

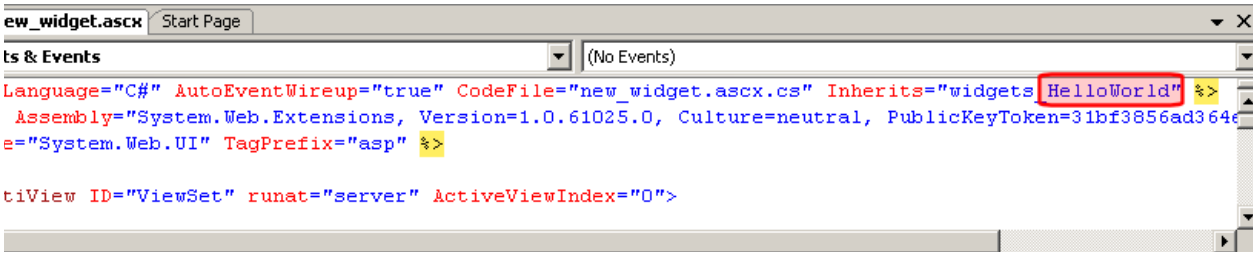
10. Copy, paste, then rename the `helloworld.ascx.jpg` file to `new_widget.ascx.jpg`. Ektron CMS400.NET administrators (like Grace) and content authors (like the Marketing team) use a widget's image to select it, as shown below.



Update the Class Names in the New Files

1. Open new_widget.ascx.

2. On the first line of that file, replace the reference to `HelloWorld` (circled below) with `new_widget`.



3. Open the codebehind file, `new_widget.ascx.cs`.

4. Again, replace the class `HelloWorld` with `new_widget`.

```
using System.Web.UI.HtmlControls;
using Ektron.Cms.Widget;
using Ektron.Cms;
using Ektron.Cms.API;
using Ektron.Cms.Common;
using Ektron.Cms.PageBuilder;
using System.Text.RegularExpressions;

public partial class widgets_HelloWorld : System.Web.UI.UserControl, IW
{
    #region properties
    private string _HelloString;
    private bool _CheckBoxBool;
    [WidgetDataMember(true)]
    public bool CheckBoxBool { get { return  CheckBoxBool; } set {  Che
```

5. Save `new_widget.ascx` and `new_widget.ascx.cs`.

6. Check both files for errors by clicking Build > Build Page. Correct any errors before proceeding.

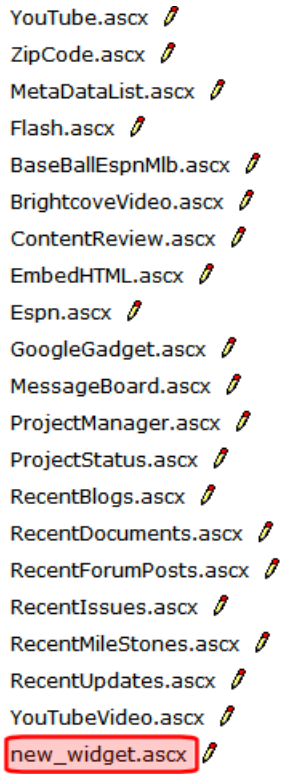
Add Widget in Ektron CMS400.NET Workarea

1. Open the Ektron CMS400.NET Workarea.

2. Open Settings > Configuration > Personalizations > Widgets.

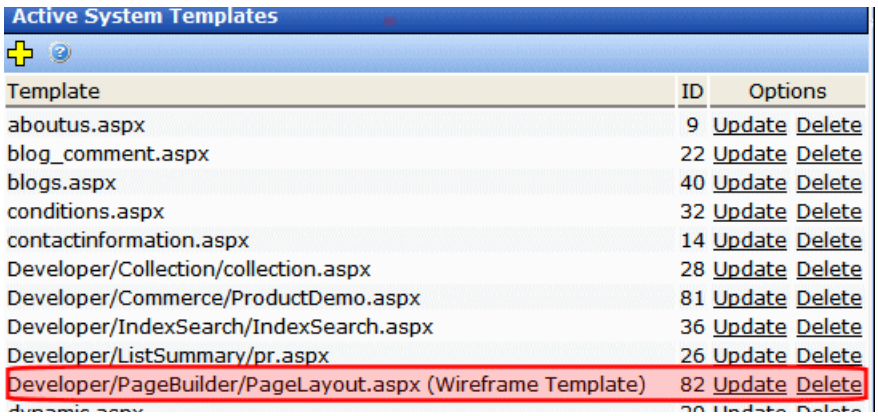
3. Click the Synchronize button ().

4. You see the new user control file, `new_widget.ascx`, at the bottom of the screen.



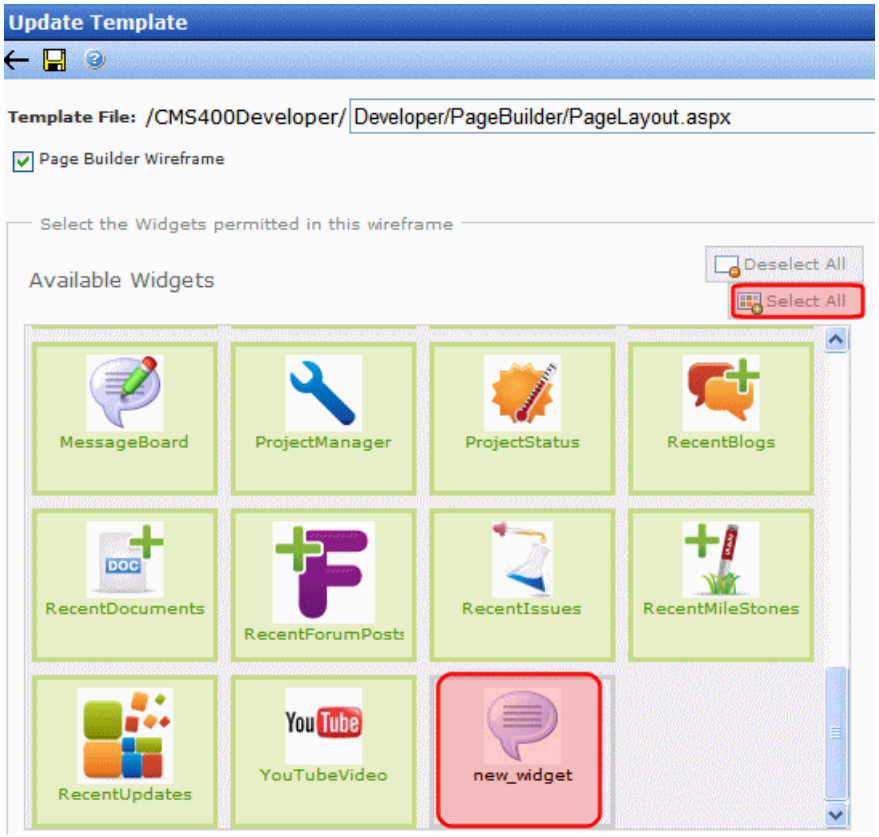
5. Open Settings > Configuration > Template Configuration.

6. Find the template that you created in "Building Pages" on page 3, PageLayout.aspx. Or, any Wireframe template that you are using to create a PageBuilder page.



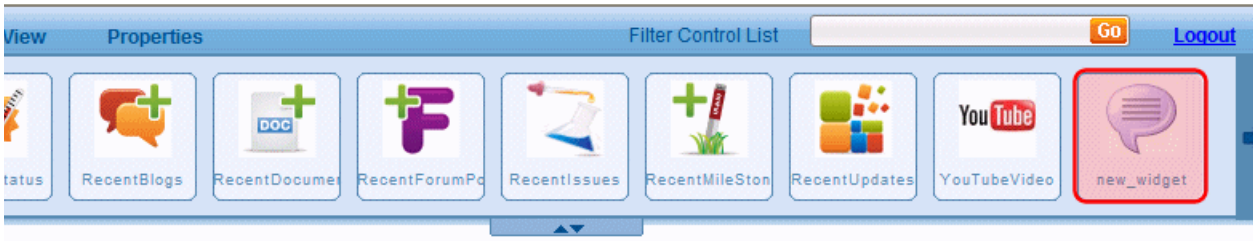
7. Click Update.

8. On the Update Template screen, scroll down until you see the new widget.



9. Click the Select All button (circled above).
10. Click Save ( ).
11. Go to Content and select a folder that has a PageBuilder page.
12. Edit the PageBuilder page.
13. Open the widget menu.
14. Make sure your new widget appears on the menu.

Note: For directions on creating a PageBuilder page, see "Steps to Creating a "PageBuilder" Page" on page 4



Now that you have created a new widget and enabled it in the Ektron CMS400.NET Workarea, you can begin to customize it.

The next sections explain details about the files you copied and renamed above.

Understanding the User Control (.ascx) File

Here is the new_widget.ascx file that Pete uses as the basis of his widget.

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="new_widget.ascx.cs" Inherits="widgets_new_widget" %>
<%@ Register Assembly="System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
    Namespace="System.Web.UI" TagPrefix="asp" %>

    <asp:MultiView ID="ViewSet" runat="server" ActiveViewIndex="0">

        <asp:View ID="View" runat="server">
            <!-- You Need To Do ..............................  -->
           <asp:Label ID="TextLabel" runat="server"></asp:Label><br />
           <asp:Label ID="CheckBoxLabel" runat="server"></asp:Label>
            <!-- End To Do ..............................  -->
        </asp:View>

        <asp:View ID="Edit" runat="server">
            <div id="<%=ClientID%>_edit">
                 <!-- You Need To Do ..............................  -->
                   <asp:TextBox ID="TextTextBox" runat="server" Style="width: 95%"> </asp:TextBox><br />
                   <asp:CheckBox ID="MyCheckBox" runat="server" Checked="false" />  <br />
                 <!-- End You Need To Do ..............................  -->
                   <asp:Button ID="CancelButton" runat="server" Text="Cancel" OnClick="CancelButton_Click" />
                   <asp:Button ID="SaveButton" runat="server" Text="Save" OnClick="SaveButton_Click" />
            </div>
        </asp:View>

    </asp:MultiView>
```

Pete notices the following elements of the file.

- The `asp:MultiView` element declares that the control has two possible modes: View and Edit.

  ```
  <asp:MultiView ID="ViewSet" runat="server" ActiveViewIndex="0">
  ```

  - in *view* mode, Pierre and the marketing team can see the control but not change it.
  - in *edit* mode, Pete's developers can change the control's content and properties.

- Between the `multiview` tags is information about the control in view mode. It has two fields: one is a text field, and the other is a check box.

  ```
  <asp:View ID="View" runat="server">

          <asp:Label ID="HelloTextLabel" runat="server"></asp:Label><br />
  ```

```
        <asp:Label ID="CheckBoxLabel" runat="server"></asp:Label>
</asp:View>
```

- Also between the `multiview` tags is information about the control in edit mode. In edit mode, a text box, a check box, and a Save button appear. The text box and check box collect end-user input, and the Save button saves that input to the database.

```
<asp:View ID="Edit" runat="server">
<div id="<%=ClientID%>_edit">
                <!-- You Need To Do ..........................  -->
        <asp:TextBox ID="HelloTextBox" runat="server" Style="width: 95%"> </asp:TextBox><br />
        <asp:CheckBox ID="MyCheckBox" runat="server" Checked="false" />  <br />
<!-- End You Need To Do ..........................  -->
        <asp:Button ID="CancelButton" runat="server" Text="Cancel" OnClick="CancelButton_Click" />
        <asp:Button ID="SaveButton" runat="server" Text="Save" OnClick="SaveButton_Click" />
```

Understanding the Codebehind (.ascx.cs) File

Next, Pete reviews the codebehind file, `new_widget.ascx.cs`.

- He see a series of `using` statements at the top of the file, noticing the Ektron ones in particular:

```
using Ektron.Cms.Widget;
using Ektron.Cms;Marketing team
using Ektron.Cms.API;
using Ektron.Cms.Common;
using Ektron.Cms.PageBuilder;
using System.Text.RegularExpressions;
```

- Pete then notes a widget host class, which inherits the `system.Web.UI.UserControl` and `IWidget` classes. *****

```
public partial class widgets_new_widget : System.Web.UI.UserControl, IWidget
```

The following image summarizes the remaining elements of the codebehind file.


```
public partial class widgets_HelloWorld : System.Web.UI.UserControl, IWidget
{
    #region properties
    private string _HelloString;
    [WidgetDataMember("Hello Wolrd")]
    public string HelloString { get { return _HelloString; } set { _HelloString = value; } }
    #endregion

    IWidgetHost _host;

    protected void Page_Init(object sender, EventArgs e)
    {   string sitepath = new CommonApi().SitePath;
        _host = Ektron.Cms.Widget.WidgetHost.GetHost(this);
        _host.Title = "Hello World Widget";
        _host.Edit += new EditDelegate(EditEvent);
        _host.Maximize += new MaximizeDelegate(delegate() { Visible = true; });
        _host.Minimize += new MinimizeDelegate(delegate() { Visible = false; });
        _host.Create += new CreateDelegate(delegate() { EditEvent(""); });
        PreRender += new EventHandler(delegate(object PreRenderSender, EventArgs Evt) { SetOutput(); });
        _host.HelpFile = sitepath + "WorkArea/help/personalization_admin.83.6.html";
        ViewSet.SetActiveView(View);
    }

    void EditEvent(string settings)
    {
        HelloTextBox.Text = HelloString;
        ViewSet.SetActiveView(Edit);
    }

    protected void SaveButton_Click(object sender, EventArgs e)
    {
        HelloString = HelloTextBox.Text;
        _host.SaveWidgetDataMembers();
        ViewSet.SetActiveView(View);
    }

    protected void SetOutput()
    {
        OutputLabel.Text = HelloString;
    }

    protected void CancelButton_Click(object sender, EventArgs e)
    {
        ViewSet.SetActiveView(View);
    }
```

- In the next line, he sees the widget's properties: a string for the text field, and a boolean for the check box. Here you define the variables and their type. Possible types are string, integer, long and date.

```
#region properties
private string _HelloString;
private bool _CheckBoxBool;
[WidgetDataMember(true)]
public bool CheckBoxBool { get { return _CheckBoxBool; } set { _CheckBoxBool = value; } }
[WidgetDataMember("Hello Wolrd")]
public string HelloString { get { return _HelloString; } set { _HelloString = value; } }
#endregion
```

- He sees a widget host declaration.

```
private IWidgetHost _host;
```

- And the widget's `page_init` events.

```
protected void Page_Init(object sender, EventArgs e)
    {
        _host = Ektron.Cms.Widget.WidgetHost.GetHost(this);
        _host.Title = "Hello World Widget";
        _host.Edit += new EditDelegate(EditEvent);
        _host.Maximize += new MaximizeDelegate(delegate() { Visible = true; });
        _host.Minimize += new MinimizeDelegate(delegate() { Visible = false; });
        _host.Create += new CreateDelegate(delegate() { EditEvent(""); });
        PreRender += new EventHandler(delegate(object PreRenderSender, EventArgs Evt) {
SetOutput(); });
        ViewSet.SetActiveView(View);
    }
```

Comments on the above code

- The `gethost` method returns a reference to the container widgethost for this widget. This is the case in both Personalization and PageBuilder.
- The `Title` property is the title of this widget. By setting it in `page_init` for the widget, we inform the host what text to put in the title bar above the widget. This works in both PageBuilder and Personalization.
- The events below `host.Title` are raised by the widgethost. It's up to the widget to subscribe to them. In all cases, if we don't subscribe to them, the icons don't show up. This is a method of attaching widget code to button clicks and other events that occur outside the widget.
- For `PreRender`: Ektron CMS400.NET renders the contents of this widget on pre-render, thus ensuring a single render event. Another option is to call `SetOutput` on the Load event, but you can only do that if the widget is not in edit mode currently.
- The final line sets the view to display mode.

- He notices the declaration of the widget's `edit` events.

```
void EditEvent(string settings)
    {
        string sitepath = new CommonApi().SitePath;
        ScriptManager.RegisterClientScriptInclude(this, this.GetType(), "widgetjavascript",
sitepath + "widgets/widgets.js");
        ScriptManager.RegisterOnSubmitStatement(this.Page, this.GetType(), "gadgetescapehtml",
"GadgetEscapeHTML('" + HelloTextBox.ClientID + "');");
        HelloTextBox.Text = HelloString;
        MyCheckBox.Checked = CheckBoxBool;
        ViewSet.SetActiveView(Edit);

}
```

Comments on the above code:

Important: You must register Javascript and cascading style sheet (css) instructions in an external file. See "Working with JavaScript and Cascading Style Sheets" on page 65

- The Edit event is triggered by the widgethost, and since Pete subscribed to it already, it calls the delegate here.
- `sitepath` is used to ensure that the correct path for included files is used across installations.
- By calling the `scriptmanager` to include the script, Pete ensures it works inside update panels. Alternatively, he can use `Ektron.Cms.Api.Js.RegisterJSInclude ScriptManager.RegisterOnSubmitStatement(this.Page, this.GetType(), "gadgetescapehtml", "GadgetEscapeHTML('" + HelloTextBox.ClientID + "');");`
- The `onsubmitstatement` is javascript that is run when the widget is submitted. It calls escape html, which cleans the submitted text to avoid any XSS.

- Pete notices the editing fields, so users can see the existing data.

```
HelloTextBox.Text = HelloString;

MyCheckBox.Checked = CheckBoxBool;

ViewSet.SetActiveView(Edit);
```

- He sees the widget's `save` events.

```
protected void SaveButton_Click(object sender, EventArgs e)
    {
        HelloString = ReplaceEncodeBrackets(HelloTextBox.Text);
        CheckBoxBool = MyCheckBox.Checked;
        _host.SaveWidgetDataMembers();
        ViewSet.SetActiveView(View);
    }
```

- He notes the widget's `SetOutput` events.

```
protected void SetOutput()
    {
       HelloTextLabel.Text = HelloString;  // client javascript remove brackets, server side adds
back
        CheckBoxLabel.Text = CheckBoxBool.ToString();
    }
```

- He notes the widget's `Cancel` events.

```
protected void CancelButton_Click(object sender, EventArgs e)
    {
        ViewSet.SetActiveView(View);
    }
```

- Finally, he reviews the encoding of the greater and less than signs.

```
protected string ReplaceEncodeBrackets(string encodetext)
    {
        encodetext = Regex.Replace(encodetext, "&lt;", "<");
        encodetext = Regex.Replace(encodetext, "&gt;", ">");
        return encodetext;
    }
```

Removing a Widget from the Workarea

At some point, the AcmeBooks.com Web team will need to remove a widget from use. To do this, Grace deletes the widget's files from the `site`

`root/widgets` folder. Next, she navigates to **Settings > Personalizations > Widgets** and clicks **Synchronize** ().

The widget is removed from the list of widgets, as well as from the Dashboard of any users and community groups.

Final Assessment from the CIO

Now, the CIO of AcmeBooks.com has a clear understanding of how PageBuilder works, and how widgets are created and placed on pages.

It's clear to her that Ektron CMS400.NET's PageBuilder will greatly enhance the company's Web presence, and allow the Marketing team to respond to changing market conditions and quickly launch and communicate promotions on the AcmeBooks.com Web site.

She is also pleased that PageBuilder will satisfy the company's Web site developers. No longer will they have to deal with numerous daily requests for content changes from Marketing. And Grace is happy because administering PageBuilder and widgets within the CMS is a simple, straight-forward process.

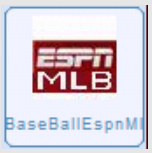
The future looks bright at AcmeBooks.com thanks to Ektron CMS400.NET and PageBuilder.

4 Creating your own Widgets

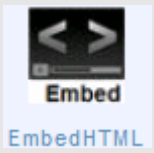
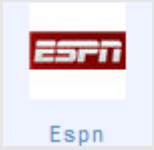
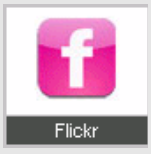
5 Standard Widgets

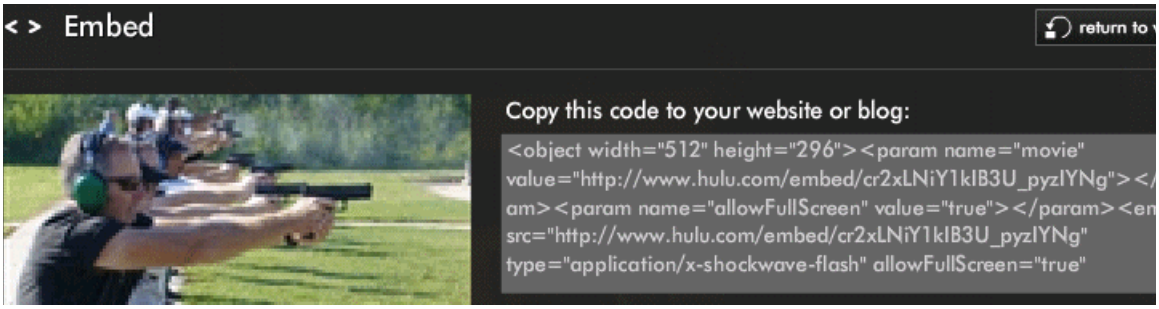
Default widgets

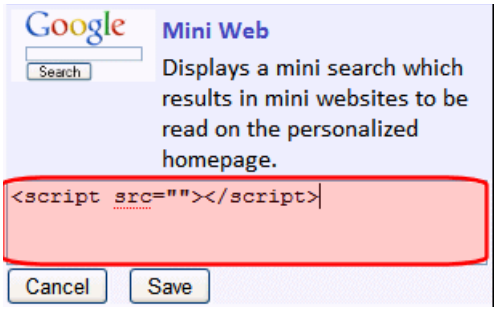
Ektron CMS400.NET installs several standard widgets to the `<webroot>/<siteroot>/Widgets/` folder. Information about them is below.

Widget	Description	Files
Atom Feed 	Lets user enter path to an Atom Publishing Protocol feed. Also lets user limit the number of feed results.	Files in widgets/Calendar and AtomFeed.ascx AtomFeed.ascx.cs atomfeed.ascx.jpg
BaseBall ESPN 	Displays latest news about the baseball team selected by editing the widget. Provided by ESPN.	BaseBallEspnMlb.ascx BaseBallEspnMlb.ascx.cs BaseBallEspnMlb.acx.jpg
Bookmarks 	Lets user create a list of URLs by entering each one's web address and title.	Bookmarks.ascx Bookmarks.ascx.cs bookmarks.ascx.jpg
BrightCove Video 	Plays any BrightCove video	Files in widgets/BrightCoveVideo and BrightCoveVideo.ascx BrightCoveVideo.ascx.cs BrightCoveVideo.ascx.jpg
Calculator 	Provides a calculator	Files in widgets/Calculator and Calculator.ascx Calculator.ascx.cs Calculator.ascx.jpg

Widget	Description	Files
Calendar 	Displays a calendar. A user can add events to any day and time.	Files in widgets/Calendar and Calendar.ascx Calendar.ascx.cs Calendar.ascx.jpg
Clock 	Provides a clock that displays time in the current time zone.	Clock.ascx Clock.ascx.cs clock.ascx.jpg
Collection 	Displays a collection. The user selects a Collection ID. The user can also define • the number entries per page • if paging is enabled • if teaser information appears • if icons are included	Collection.ascx Collection.ascx.cs Collection.ascx.jpg
ContentBlock 	Lets user enter a content ID and display that content in the Widget. Alternatively, user can create new HTML content from the widget.	Files in widgets/ContentBlock ContentBlock.ascx ContentBlock.ascx.cs contentblock.ascx.jpg
Content List 	Displays a list of content blocks. In contrast to a List Summary, where content must be in a specified folder, the ContentList control displays content from any Ektron CMS400.NET folder. The user can also define • if teaser information appears • if icons are included • the content's sort criterion (such as title, last modified date) • Sort direction	ContentList.ascx ContentList.ascx.cs ContentList.ascx.jpg
Content Review 	Places a ContentReview server control on the page. This control allows site visitors to rate and review the current page. For more information, see the ContentReview Server Control in the *Developer Guide*.	Files in widgets/ContentReview ContentReview.ascx ContentReview.ascx.cs ContentReview.ascx.jpg

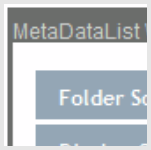
Widget	Description	Files
EmbedHTML 	Inserts a browser plugin into your page. The <embed> tag also lets you determine information about the plugin, such as: • source of plugin to be played • width • height • allow full screen (boolean) • allow script access (boolean) Examples of sites that provide embed code are YouTube.com, BrightCove, and hulu.com. See example from hulu.com below.	EmbedHTML.ascx EmbedHTML.ascx.cs EmbedHTML.ascx.jpg
ESPN 	Displays selected ESPN news feed. To select a feed, click Edit button (circled below) and click a feed. 	espn.ascx espn.ascx.cs espn.ascx.jpg
Flash 	Displays a selected flash file which resides in Ektron CMS400.NET. You can also set the display's height and width. See Also: "Working with the Flash Widget" on page 56	Files in widgets/Flash Flash.ascx Flash.ascx.cs Flash.ascx.jpg
Flickr 	Display Flickr's Most Recent or Most Interesting photos. The user also can select the number of rows and columns for the image display. See Also: "Working with the Flash Widget" on page 56	Flickr.ascx Flickr.ascx.cs flickr.ascx.jpg



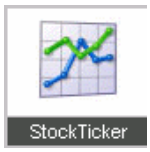
Widget	Description	Files
GoogleGadget 	Gadgets designed for an IGoogle page. By default, Ektron supplies 5 Google gadgets. To insert another Google gadget, follow these steps. **1.** Drop a Google gadget widget. **2.** Click the pencil to enter edit mode. **3.** Open a new tab. **4.** Go to http://www.google.com/ig/directory?synd=open&hl=en-US&gl=US&cat=all. **5.** Find the gadget you want to insert. **6.** Click Add to your Web page. **7.** Click Get the code. **8.** Copy the code. **9.** Return to the PageBuilder page. **10.** Paste the code into the bottom of the Google widget you dropped in Step **1**.  Click Save.	Files in widgets/GoogleGadget GoogleGadget.ascx GoogleGadget.ascx.cs GoogleGadget.ascx.jpg
HelloWorld 	Very simple widget. Created by Ektron to help developers understand how to create their own widgets.	HelloWorld.ascx HelloWorld.ascx.cs HelloWorld.ascx.jpg

Widget	Description	Files
iFrame 	Lets user enter a path to a Web page or an item on the Web page.	IFrame.ascx IFrame.ascx.cs iframe.ascx.jpg
List Summary 	Displays an Ektron List Summary, a list of certain types of content in a selected folder. Optionally, the display can include • number of items to display on the page • if the number of items in a folder exceeds the number to display, should you display the rest on additional pages? • if icons are included • the content's sort criterion (such as title, last modified date) • sort direction • an override of the Add Content menu item's default text. For example, for a news site, you could change Add Content to Add News. • displays in the widget content identified by the current query string parameter. A List Summary widget displays the following types of content. • HTML content • PageBuilder page • XML Smart Form • Blog This widget does *not* display the following content types. • Catalog entries (part of the eCommerce feature) • Forums • HTML Forms • Assets	Files in widgets/ListSummary ListSummary.ascx ListSummary.ascx.cs ListSummary.ascx.jpg
MessageBoard 	Allows user to leave comments on the page. For a description of the widget's properties, see the MessageBoard Server Control in the *Developer Guide*.	Files in widgets/MessageBoard MessageBoard.ascx MessageBoard.ascx.cs MessageBoard.ascx.jpg

Widget	Description	Files

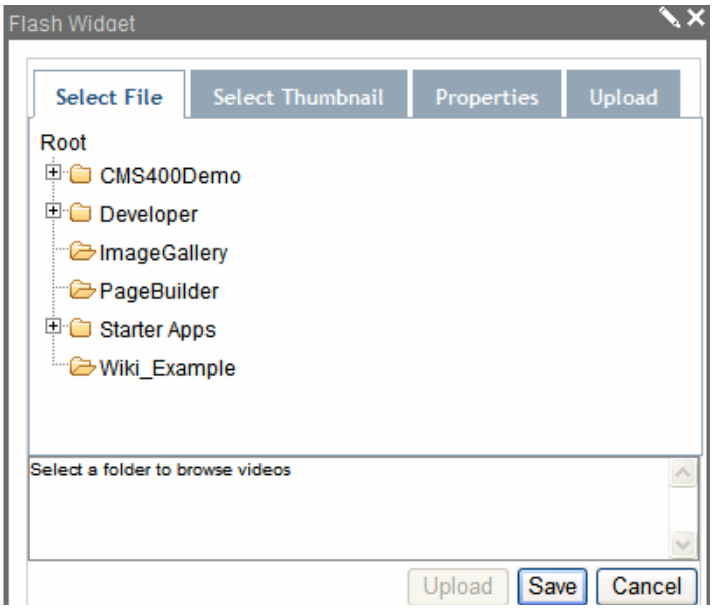
Metadata List
 
	Displays content whose metadata fits a selected folder location and keywords.
	You can also set these display options:
	• number of items to display on the page
	• if the number of items in the folder exceeds number to display, should remaining items appear on additional pages?
	• Navigation/Teaser: Determines display of content
	  ❍ ecmNavigation - lists title of every content item
	  ❍ ecmTeaser - lists title and summary
	  ❍ ecmUnOrderedList - lists title and summary; unsorted list
	• if icons are included
	• the content's sort criterion (such as title, last modified date)
	• sort direction
		Files in widgets/MetaDataList
		MetaDataList.ascx
		MetaDataList.ascx.cs
		MetaDataList.ascx.jpg

News 	Lets user select a news feed from a group of major news providers. Note: A developer can change the list of news feeds by editing the `siteroot/widgets/news.ascx.cs` file.	IFrame.ascx IFrame.ascx.cs iframe.ascx.jpg
Recent blog posts 	Displays a selected number of the most recent blog posts.	RecentBlogs.ascx RecentBlogs.ascx.cs RecentBlogs.ascx.jpg
Recent documents 	Displays a selected number of the most recently published documents.	RecentDocuments.ascx RecentDocuments.ascx.cs RecentDocuments.jpg
Recent forum posts 	Displays a selected number of the most recent forum posts.	RecentForumPosts.ascx RecentForumPosts.ascx.cs RecentForumPosts.ascx.jpg

Widget	Description	Files
RSS Feed 	Allows a user to enter the path of a feed that uses Really Simple Syndication (RSS).	RSSFeed.ascx RSSFeed.ascx.cs rssfeed.ascx.jpg
Sales Force Chart 	Allows users to enter username, password and information about a Sales Force chart to display that chart.	SalesForceChart.ascx SalesForceChart.ascx.cs SalesForceChart.ascx.jpg
Stock Ticker 	Allows users to define a list of stock ticker symbols and display the price for each symbol.	StockTicker.ascx StockTicker.ascx.cs StockTicker.ascx.jpg StockTicker.css
Text Box 	Displays a single field to capture text. The text is stored in the database.	TextBox.ascx TextBox.ascx.cs TextBox.ascx.jpg
Weather 	Allows the user who is dropping the widget to enter a Zip Code, then displays its weather information.	ZipCode.ascx ZipCode.ascx.cs ZipCode.ascx.jpg
YouTube 	Allows users to select from a list of YouTube feeds. When the page appears, videos in the category appear. Note: A developer can change the list of youtube feeds by editing the `siteroot/widgets/youtube.ascx.cs` file.	YouTube.ascx YouTube.ascx.cs YouTube.ascx.jpg

Widget	Description	Files
You Tube Video 	Allows user to embed code for any YouTube video.	YouTubeVideo.ascx YouTubeVideo.ascx.cs YouTubeVideo.ascx.jpg
Zip Code 	Allows users to find the information shown below about a location by entering one of the items. • Zip Code • Area Code • State • City	ZipCode.ascx ZipCode.ascx.cs ZipCode.ascx.jpg

Working with the Flash Widget



- This widget supports both .flv and .swf file types.

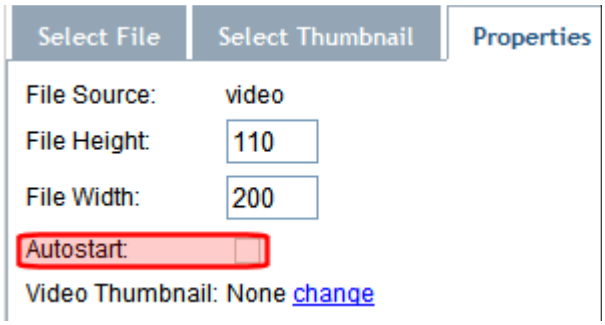
  If you will use .flv files, make sure it is added as a file type. See the *Administrator Guide* for more information.

- If the Flash file you want to display *has* already been added to Ektron CMS400.NET, use the Select File tab to it.

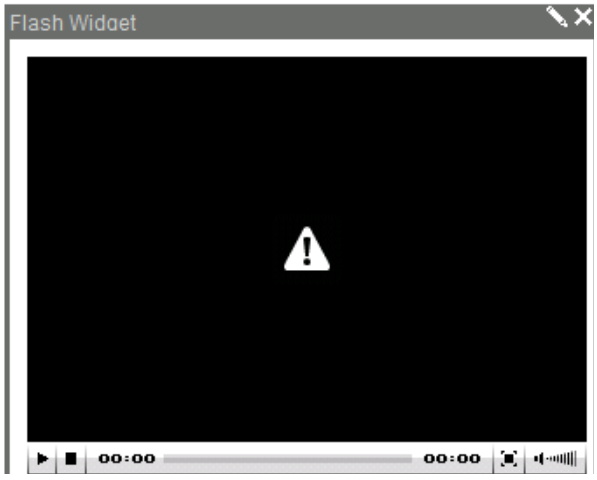
  If the file has *not* been added to Ektron CMS400.NET, first use the Select File tab to select a folder for the Flash file. Then, use the Upload tab to navigate your computer's file system, and upload the file to the selected folder.

- The thumbnail feature only works with .flv file types. If you select a Thumbnail, it appears within the Flash player when the page loads. See example below.

- The thumbnail is only applied to this instance of the widget - not the Flash file. If you apply this video to a different widget on this page, you must reapply a thumbnail.
- The thumbnail is an image file that was dropped into Ektron CMS400.NET as an asset. It cannot be a library image.
- The autostart feature only works with .flv files.



- If you log in then upload a flash file, and certain requirements are not met for that file, you see the following image where the widget appears.



The following conditions cause this image to appear.

- The flash file's folder properties require certain metadata and/or a taxonomy category to be applied, and they have not.
- The folder has an approval chain, and this content has not been approved.

6 Advanced PageBuilder Topics

Getting the most out of PageBuilder

ektron CMS400.net

This section covers the following advanced topics that help you implement PageBuilder more fully.

Wireframes:

Widgets:

Customizing the PageBuilder Menu Control

You can customize the PageBuilder Menu user control's behavior in the following ways.

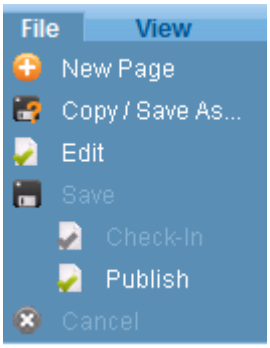
Determining the Ektron CMS400.NET Folder to Which Pages are Saved

How a Page's Default Folder is Set

By default, when a site user creates a new PageBuilder page from an existing one, it is saved to the same Ektron CMS400.NET folder. For example, consider this folder/content structure.

```
Root (folder id 0)

Products (folder id 20)

        PageLayout: “Omaha Mailorder Steaks” (content id 35)

Services (folder id 30)
```

If you are viewing `content id` 35 and click New Page or Copy/Save as, the new page is saved to that content's folder (`Products, folder id= 20`).



Note: When you create a new page layout from the *Workarea*, you first select a folder then begin creating the page layout.

Overriding the Default Folder

Use the PageBuilder control's `FolderID` property to override the default and specify an Ektron CMS400.NET folder to which new pages are saved. Here is an example of that property:

```
<ucPageBuilder:PageHost ID="ucPageHost1" FolderID="25" runat="server" />
```

When a user is working on a page that hosts this PageBuilder control, and he saves a new page, it is saved to `folder id` 25.

Changing the Page's Cache Interval

Use the `CacheInterval` property to cache a page and its Widgets that represent Ektron CMS400.NET server controls, such as a Collection Widget.

This property sets the amount of time, in seconds, that data is cached. The default is 0 (zero).

```
<ucPageBuilder:PageHost ID="ucPageHost1" CacheInterval="25" runat="server" />
```

Customizing the DropZone User Control

Three properties lets you customize the DropZone user control's behavior in the following ways.

- "Letting Users Add Columns to a DropZone" on page 60
- "Letting Users Resize a Dropzone" on page 61
- "Setting a DropZone's Column Widths Programmatically" on page 61

Letting Users Add Columns to a DropZone

Use the `AllowAddColumn` property to let users add columns to a DropZone.

```
<ucPageBuilder:DropZone ID="ucDropZone1" AllowAddColumn="true" AllowColumnResize="true"
runat="server" />
```

For example, a Dropzone initially contains one column, but the page creator wants three. If this property is set to `true`, this change is possible.

This property's default value is `true`.

Letting Users Resize a Dropzone

Use the `AllowColumnResize` property to let users change the width of columns in a DropZone.

```
<ucPageBuilder:DropZone ID="ucDropZone1" AllowAddColumn="true" AllowColumnResize="true"
runat="server" />
```

For example, column width is 100% by default. A page creator wants to change it to 50%. If this property is set to `true`, this change is possible.

This property's default value is `true`.

Setting a DropZone's Column Widths Programmatically

If you want to set a dropzone's column widths programmatically, follow these steps.

1. Add the following `Register` statement to the page's `<head>` tags.

```
<%@ Register Assembly="Ektron.Cms.Widget" Namespace="Ektron.Cms.PageBuilder" TagPrefix="PB" %>

<%@ Register Src="~/Workarea/PageBuilder/PageControls/PageHost.ascx" TagPrefix="PB"
TagName="PageHost" %>

<%@ Register Src="~/Workarea/PageBuilder/PageControls/DropZone.ascx" TagPrefix="PB"
TagName="DropZone" %>

<%@ Register Assembly="Ektron.Cms.Widget" Namespace="Ektron.Cms.PageBuilder" TagPrefix="PB"%>
```

Important: The Widget and Dropzone assemblies must have the same Tag Prefix. See example.

2. Within the page's `<body>` tags, add the following code. the following example adds three columns of 100 pixels each.

```
<PB:DropZone ID="Middle" runat="server">

<ColumnDefinitions>

<PB:ColumnData width="100" columnID="0" unit="pixels"></PB:ColumnData>

<PB:ColumnData width="100" columnID="1" unit="pixels"></PB:ColumnData>

<PB:ColumnData width="100" columnID="2" unit="pixels"></PB:ColumnData>

</ColumnDefinitions>

</PB:DropZone>
```

Set `width` to an appropriate number. For the `unit`, the options are

- pixels
- percent
- em

If you set a dropzone's column widths programmatically, you must also set the `AllowAddColumn` and `AllowColumnResize` properties to `false`. If you do not, users working with widgets can add columns and adjust column widths on the page, but their changes will revert to these settings when they try to save.

Customizing the Wireframe

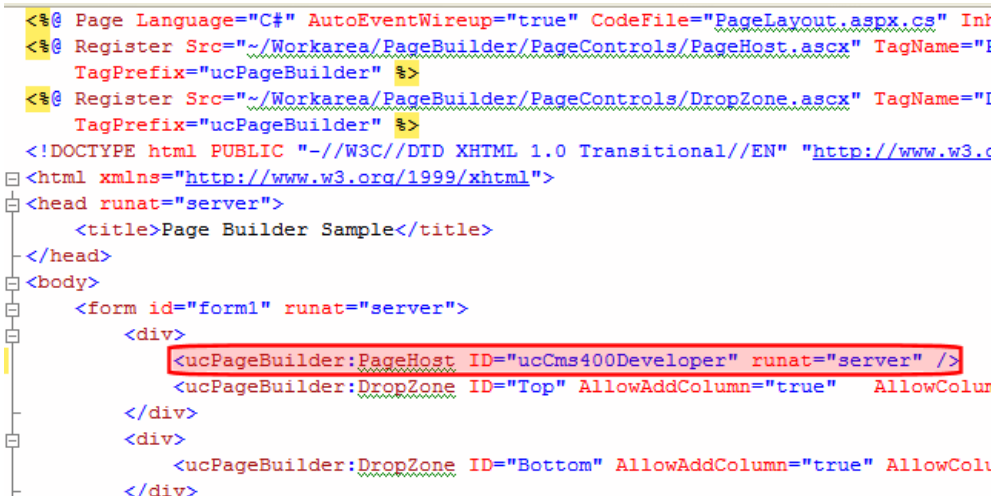
Assigning a Default Page to a Wireframe

You can assign a default page to a wireframe. If you do, and a site visitor enters a URL with a path to that wireframe that lacks a query string ID, the default page appears. The following example explains this feature.

URL	Returns this PageLayout content
`http://site root/cms400developer/developer/PageBuilder/PageLayout.aspx?pageid=1036`	In the PageBuilder folder, PageLayout ID 1036
`http://site root/cms400developer/developer/PageBuilder/PageLayout.aspx` (Note lack of query string parameter)	The PageLayout page identified in the `PageLayout.aspx` file's PageBuilder menu user control `DefaultPageID` property.

To assign a default page to a wireframe, follow these steps.

1. In the Ektron CMS400.NET Workarea, create a PageBuilder page that will be used as the default pageid for a wireframe.
2. In Visual Studio, open the wireframe file assigned to that folder.
3. Find the PageBuilder menu user control (circled below).



4. Add a new property, `DefaultPageID`.
5. For the property's value, enter the ID of the page you created in Step 1.

Here is an example of that line with the `DefaultPageID` property added.

```
<ucPageBuilder:PageHost ID="ucCms400Developer" DefaultPageID="1035" runat="server" />
```

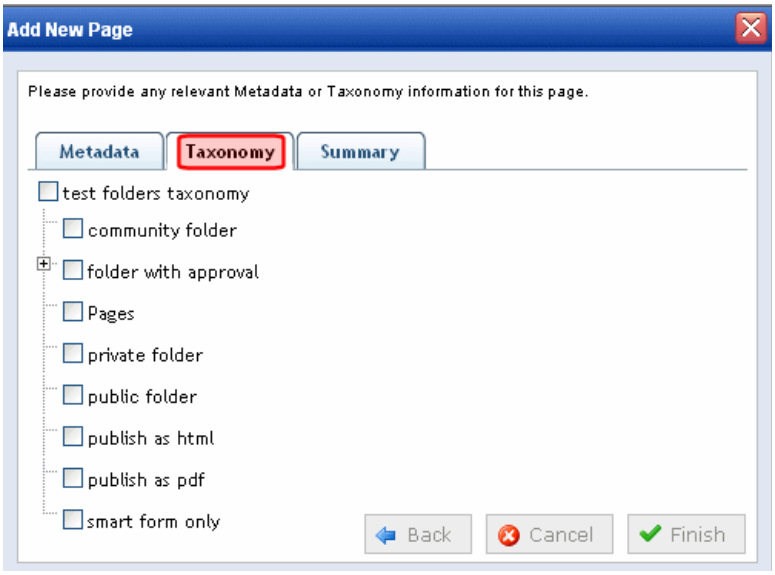
6. Save your changes.

To continue the above example, if someone opens a browser and enters `http://site root/cms400developer/developer/PageBuilder/PageLayout.aspx`, he is redirected to

`http://site root/cms400developer/developer/PageBuilder/PageLayout.aspx?pageid=1035`

Assigning a Default Taxonomy to a Wireframe

While creating a new PageBuilder page in the Ektron CMS400.NET Workarea, the user can assign one or more of the taxonomy categories that are set in the page's folder properties. See example below.



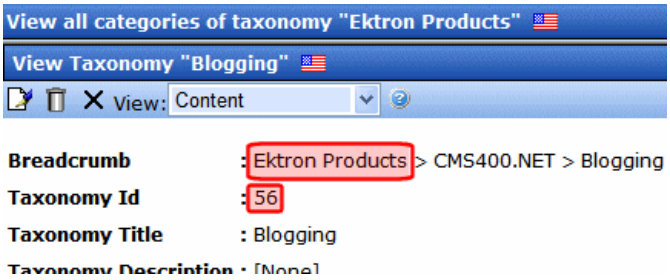
As a developer, you can assign a *default* taxonomy category to a wireframe. If you do, and the user creating a page using that wireframe makes no changes, the default category is assigned to the page. However, the user can change the taxonomy when the Add New Page screen appears.

Note: Default taxonomies are applied only when a user logs into a Web site and adds a new page -- they are *not* applied when creating new pages within the Workarea.

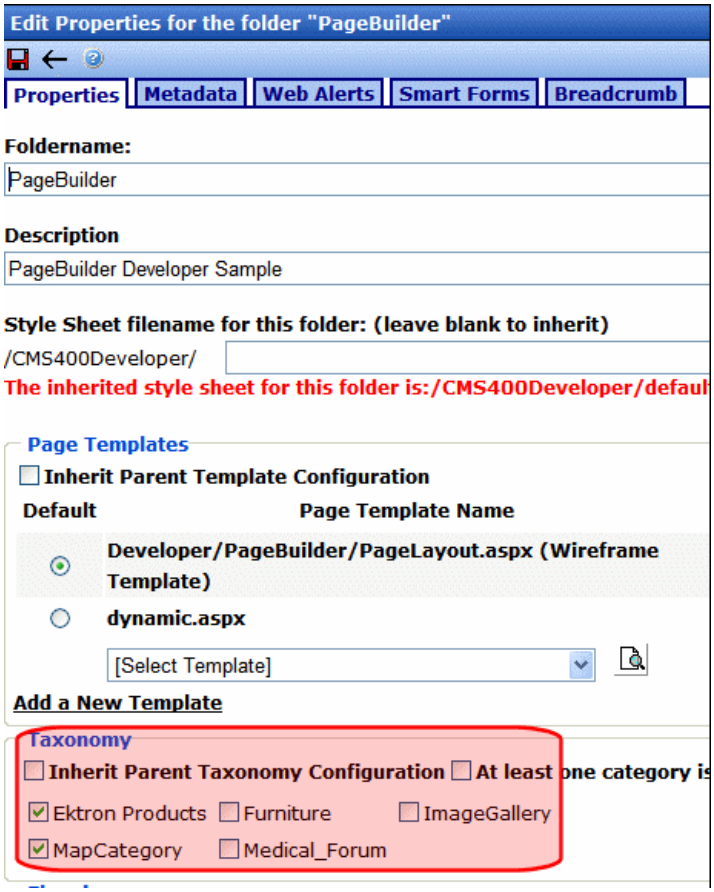
To assign a default taxonomy category to a wireframe, follow these steps.

Prerequisite

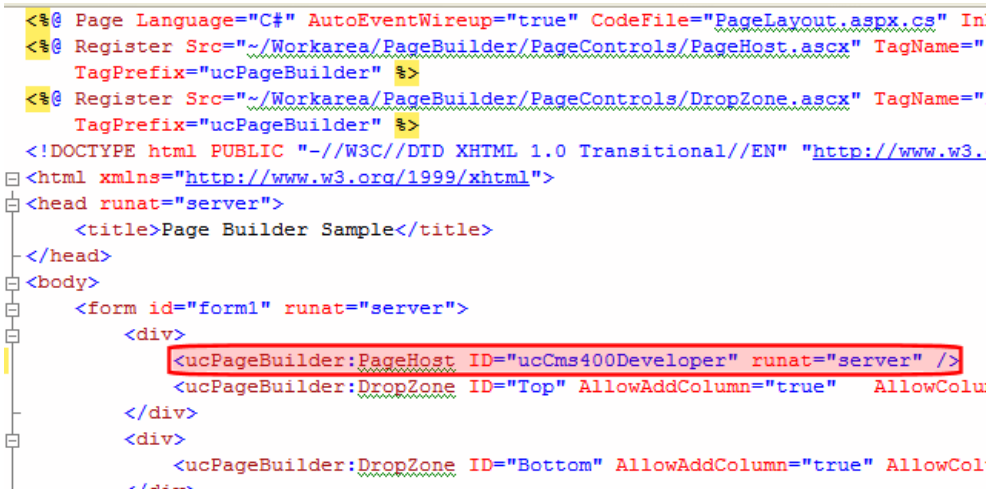
The ID number and the name of the top-level parent taxonomy for the default taxonomy category. For example, the screen below shows that the Blogging category's ID is 56, and its parent taxonomy is Ektron Products.



1. In the Ektron CMS400.NET Workarea, navigate to the folder properties screen of the folder to which the wireframe is applied.
2. Click the Edit button ( ).
3. Locate the Taxonomy area of the Edit Folder properties screen.



4. If you will assign to the wireframe a top-level taxonomy (that is, one of those in the area circled above), check its box.
5. If you will assign a taxonomy *category* (that is, a child node below a top-level taxonomy), check the box of its *parent* Taxonomy.
6. Save your changes to folder properties.
7. Open Visual Studio.
8. Open the wireframe to which you will assign a default taxonomy.
9. Find the PageBuilder menu user control (circled below).



10. Add a new property, `SelTaxonomyID`.

11. For the property's value, enter the ID of the default taxonomy or category.

Here is an example of that control with the `SelTaxonomyID` property added.

```
<ucPageBuilder:PageHost ID="ucCms400Developer" SelTaxonomyID=”13” runat="server" />
```

12. Save your changes.

To continue the above example, the next time someone creates a page based on that wireframe, taxonomy ID 13 will be the page's default taxonomy category. If desired, the user can change it by navigating to the Taxonomy tab and assigning different or additional categories.

Customizing Widgets

The following topics let you further customize widget behavior.

Working with JavaScript and Cascading Style Sheets

You can use JavaScript or a cascading style sheet to add custom functionality or styling to a widget. To do this, place the JavaScript or cascading style sheet (css) instructions in an external file, then register it in the codebehind file.

Here is an example of including a JavaScript file.

```
void EditEvent(string settings)

JS.RegisterJSInclude(this, _api.SitePath + "widgets/contentblock/jquery.cluetip.js",
"EktronJqueryCluetipJS");
```

Here is an example of including a .css file.

```
Css.RegisterCss(this, _api.SitePath + "widgets/contentblock/CBStyle.css","CBWidgetCSS");
```

Important: You must register JavaScript and .css files in an external file, as shown above. If you do not, the OnSubmit event places HTML in the TextArea field in encoded brackets (< >) and generates a dangerous script error.

The `JS.RegisterJSInclude` and `Css.RegisterCss` functions take three arguments.

Argument	Description	Example (from above code)
1	A reference to the control that needs the script or style sheet on the page. Typically, 'this' or 'me'.	`this`
2	A unique key. Only include the script specified by a key once. Your organization should develop a standard way to define JavaScript and .css keys.	• `"EktronJqueryCluetipJS"` • `"CBWidgetCSS"`
3	The URL of the script or style sheet being included. Ektron recommends prefixing the URL with a sitepath, so it can be used with URLs like `http://localhost/ektrontech` and `http://ektrontech`.	• `_api.SitePath + "widgets/contentblock/jquery.cluetip.js"` • `_api.SitePath + "widgets/contentblock/CBStyle.css"`

Note that Widgets use an update panel for partial postbacks. As a result, the ASP.NET tree view and file upload controls do not work with widgets. Ektron CMS400.NET has workarounds for these functions. For an example of a tree view, see the content block widget (*site root*/widgets/contentblock.ascx). For an Ajax file uploader, see the flash widget (*site root*/widgets/flash.ascx).

Verifying that a Page is a PageBuilder Page

Whenever your code is interacting with a widget, you need to verify that it is on a page builder page (as opposed to another Ektron CMS400.NET page that hosts widgets, such as personalization).

To check for this, insert the following code:

```
Ektron.Cms.PageBuilder.PageBuilder p = (Page as PageBuilder);

If(p==null) // then this is not a wireframe
```

When you want to check the mode, use code like this.

```
If(p.status == Mode.Edit) // we are in edit mode
```

Applying Global and Local Properties to Widgets

Global and local widget properties reduce your development effort by eliminating settings data classes. While you can still use these classes and manage your own serialization, for the vast majority of types, the built-in engine performs all the work necessary.

Global properties apply to every instance of a widget. *Local* properties can apply to one instance of a widget. If both local and global values are assigned to a property, the local overrides the global.

As an example of using a local property to override a global, consider a ListSummary widget. You may want its sort to mostly be by modified date in descending order, but in certain instances you want to override that and sort by title in ascending order.

The following table explains how to set the different property types.

Type	Setting default value in codebehind file	How to change default
Global	`[GlobalWidgetData()]` `public string NewWidgetTextData { get { return _NewWidgetTextData; } set { _NewWidgetTextData = value; } }`	Workarea > Settings > Configuration > Personalizations > Widgets
Local	`[WidgetDataMember()]` `public string NewWidgetTextData { get { return _NewWidgetTextData; } set { _NewWidgetTextData = value; } }`	User drops widget onto PageBuilder page, then clicks edit button ()

See Also: "Setting a Widget's Global Properties" on page 67; "Setting a Widget's Local Properties" on page 69

Setting a Widget's Global Properties

A *global* property lets an Ektron CMS400.NET developer or administrator assign properties and values that apply to all instances of a widget. You apply a global property to the widget's codebehind page. Administrators could then set or update the property's value in the Workarea's Widgets screen.

For example, the BrightCove Video widget requires a player ID. You could insert that in the widget's codebehind file. Then, an administrator could review and possibly update that information in the Workarea widgets screen. Whenever a user drops a BrightCove Video widget onto a page, the player ID is already assigned.

If the developer does *not* set a default value in the codebehind, an administrator must set one on the Workarea's Widgets screen.

If the developer *does* set a default value in the codebehind, it will be applied unless changed by an administrator on the Workarea's Widgets screen.

Steps for Setting a Global Property

Follow these steps to set a global property.

1. Open the widget's codebehind file, which is located in the *site root*`/widgets` folder.
2. In the `properties` section, insert the `GlobalWidgetData` attribute (shown below) to set the global property's name and type.

```
[GlobalWidgetData()]

public string NewWidgetTextData { get { return _NewWidgetTextData; } set { _NewWidgetTextData = value; } }
```

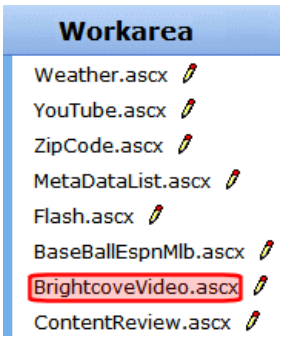
Here is an example.

```
public partial class Widgets_BrightcoveVideo : System.Web.UI.UserControl, IWidget
{
    #region properties
    private long _PlayerID;
    public long _VideoID;
    [WidgetDataMember(15037095001)]
    public long PlayerID { get { return  PlayerID; } set {  PlayerID = value; } }
    [GlobalWidgetData(15053010001)]
    public long VideoID { get { return _VideoID; } set { _VideoID = value; } }
    public long publisherID = 14459838001;  // Your company Publisher ID Get from
    public string token = "R2tftyEmD8Kn3M3zoXFj9gA7rax2BIzZDijTcsUCkJl5tXA8c-hgwQ
    #endregion
```

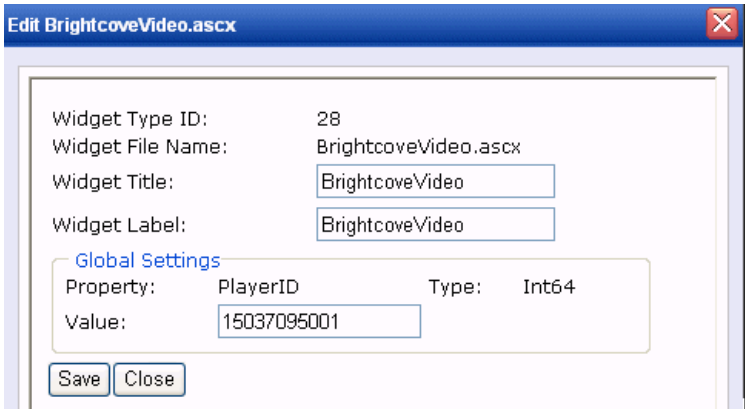
The supported types for `GlobalWidgetData` are

- Date Time
- int
- long
- double
- boolean
- string
- any enumeration

1. Save the codebehind file.
2. In the Ektron CMS400.NET Workarea, go to Settings > Configuration > Widgets.
3. Click the edit icon ( ) for the widget whose codebehind file you edited in Step 1.



4. A dialog lets you view and edit global properties set in the codebehind file. See example below.



Setting a Widget's Local Properties

A *local* property lets an Ektron CMS400.NET user assign property values that apply to a particular instance of a widget. For example, the BrightCove Video widget requires a Video ID, which identifies the video that appears where you drop the widget.

To set a local property, follow these steps.

1. Open the widget's codebehind file, which is located in the *site root*`/widgets` folder.
2. In the `properties` section, insert the `WidgetDataMember` attribute to set the property. See example below.

```
[WidgetDataMember(150530105432)]1

public long VideoID { get { return _VideoID; } set { _VideoID = value; } }
```

3. If you want to set a default value for the widget, use the attribute's optional argument, which follows `[WidgetDataMember`. In the example above, the value is `150530105432`.
4. Save the settings in your properties by populating them as you normally would.
5. In the Save event, call `_host.SaveWidgetDataMembers();`. See example below.

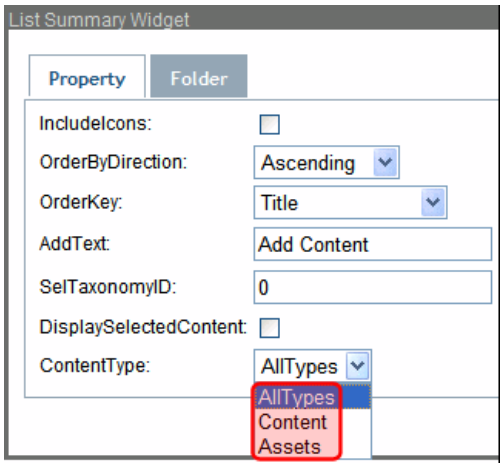
```
protected void SaveButton_Click(object sender, EventArgs e)
{
      VideoID = Int64.Parse(tbData.Text);
     _host.SaveWidgetDataMembers();
     ViewSet.SetActiveView(View);
}
```

Adding a Field to a Widget

This section provides an example of adding a Content type drop down to the List Summary widget. The drop down lets the person dropping the widget on the page select from these choices.

- all types of content
- HTML content only
- assets only

Here is what the drop down looks like once it is implemented.



To add this drop down to the List Summary widget, follow these steps.

1. In Visual Studio, open the ListSummary widget, *site root*`/widgets/ListSummary.ascx`.
2. Find the text `DisplaySelectedContent`.
3. Below `DisplaySelectedContent`, add the following code to create a drop down list for the `ContentType` property.

```
<tr>
        <td>
                    DisplaySelectedContent:</td>
        <td>
                    <asp:CheckBox ID="DisplaySelectedContentCheckBox" runat="server" />
        </td>
</tr>
<tr>
```

```
        <td>
                ContentType:
        </td>
        <td>
                                <asp:DropDownList ID="ContentTypeList" runat="server">
                                <asp:ListItem Value="AllTypes">AllTypes</asp:ListItem>
                                <asp:ListItem Value="Content">Content</asp:ListItem>
                                <asp:ListItem Value="Assets">Assets</asp:ListItem>
                                </asp:DropDownList>
        </td>
 </tr>
```

4. Save the ListSummary.ascx file.

5. Open the codebehind file, `ListSummary.ascx.cs`.

6. In the properties region, declare a string variable for the `ContentType` property, as shown below.

```
private string _ContentType;
```

7. Create a local property with default setting of `AllTypes`, as shown below.

```
[WidgetDataMember("AllTypes")]
    public string ContentType { get { return _ContentType; } set { _ContentType = value; } }
```

8. In the `EditEvent` area, set the select list's value to `ContentType`.

```
ContentTypeList.SelectedValue = ContentType;
```

9. In the `SaveButton_Click` event, set `ContentType` as the select list's value.

```
ContentType = ContentTypeList.SelectedValue;
```

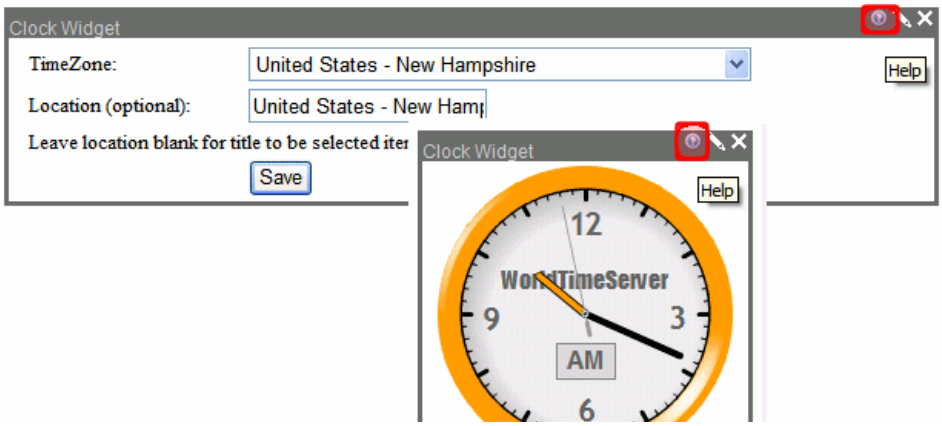
10. In the `SetListSummary()` function, set the List Summary server control's `ContentType` to the `CMSContentType` property.

```
ListSummary1.ContentType = (CMSContentType)Enum.Parse(typeof(CMSContentType), ContentType);
```

11. Save the ListSummary.ascx.cs file.

Including Help for a Widget

You can include help for any widget. The help is intended to guide the user who is dropping the widget on the page and setting its properties.



The help icon only appears when a user is editing a PageBuilder page. The icon appears both when a user is viewing a widget and editing its properties.

It is not available to a page's site visitors.

Defining a Widget's Help File

To create a widget's help file, follow these steps.

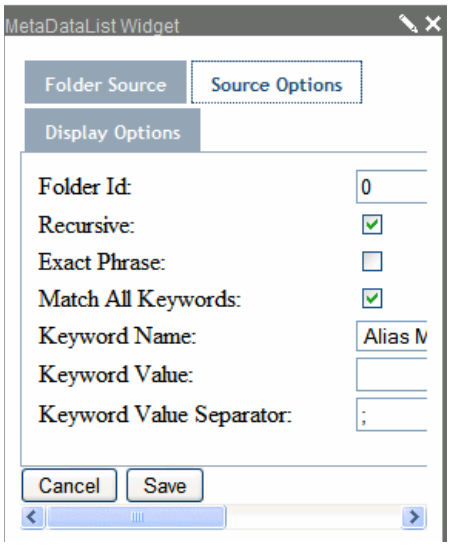
1. Create an HTML file with information for users who will drop the widget on the PageBuilder page.
2. Save the help file to the folder that contains the widget.
3. Add the WidgetHost's `HelpFile` property to the codebehind of the page that hosts the widget. See example below.

Note: You could create a content block within Ektron CMS400.NET then switch to source view, copy the content into a word processor (like Notepad), and save it with an HTML extension.

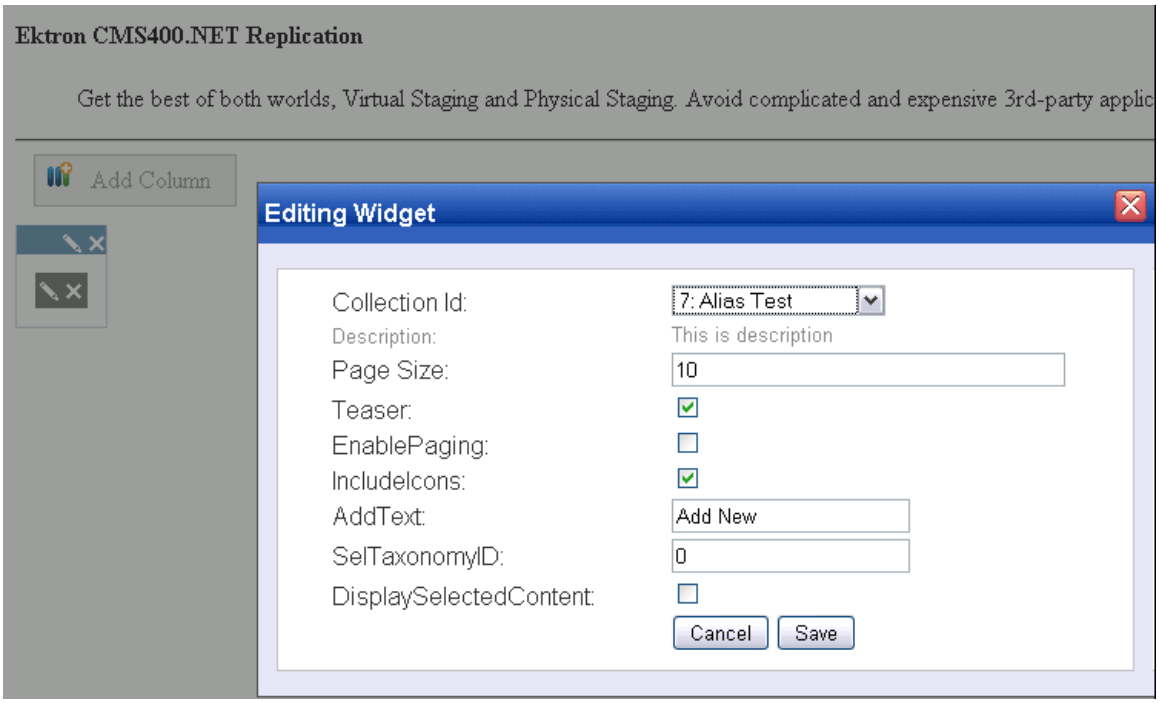
```
protected void Page_Init(object sender, EventArgs e)

{

_host = Ektron.Cms.Widget.WidgetHost.GetHost(this);

_host.HelpFile = "~/widgets/myWidget/help.html";
```

Opening a Widget's Edit Properties Screen in a Modal Dialog

If your PageBuilder page has several columns, some property screens do not have adequate space for data entry. See example below.



If desired, you can display a widget's properties screen in a modal dialog, which is much wider and provides room for data entry. See example below.



To do this, use the `ExpandOptions` property in the widget's codebehind file. Its type is `Ektron.Cms.Widget.Expandable`, and it is an enumeration with three possible values.

To have the edit button	Use this option
Open a modal dialog, where the user can edit properties	Within the widget's `page_init` event, set `_host.ExpandOptions = Expandable.ExpandOnEdit;`

Provide a button that a user can click to open a modal dialog	1. Within the widget's `page_init` event, set
	`_host.ExpandOptions = Expandable.DontExpand;`
	2. In your EditEvent callback, update the property to
	`_host.ExpandOptions = Expandable.ExpandOnExpand;`
	See "Option to Display Properties Screen as Modal Dialog" on page 74.

Show edit options in a regular window Note: This is the default setting. You can do nothing to achieve this result.	Within the widget's `page_init` event, set `_host.ExpandOptions = Expandable.DontExpand;`

Option to Display Properties Screen as Modal Dialog