CrownPeak™

TECHNICAL DOCUMENTATION

TEMPLATE DEVELOPMENT ADVANCED WORKSHOP: WIDGETS

July 2013

TABLE OF CONTENTS

IMPLEMENTING WIDGETS ON CROWNPEAK

↗ Rationale

When building web pages, you often have content that you want to include on multiple pages in your site. An example of this might be a contact information box that appears as an inset on a page, or some other callout box that you want to show on multiple pages.
In CrownPeak, you can create sub assets or widgets that allow you to edit and maintain these sub-pages of content in one place for consistency and ease of deployment but include them on multiple templates. You can even create pages that are primarily built from widgets and/or widget templates, a design pattern that can work well for modular pages and/or home pages.

↗ Assumptions

We're assuming you've already completed CrownPeak Developer training and/or the online Template Development Workshop and have basic familiarity with the CrownPeak API and Development tools, such as CrownPeak Desktop Connect (CDC) and access to a CMS for development.

↗ Objectives

- Create a widget template than can be used for managing and publishing shared content
- Create a model for creating these widgets and a site subfolder for storing widgets
- Add a document selection field to an existing template
- Load and display an external asset with the `.show()` API call
- Learn different options for publishing widgets either as part of the parent asset or as a separate asset [some details may be out of scope for different platforms].

↗ Lesson 1: Create a new template sub folder and container template

Your client, Advent General, has requested that a contact list be added in the CMS such a way that it can be edited/maintained in one place and included on a few hundred article pages in a section of online product information articles.

1. Log into the training instance: https://cms.crownpeak.com/Training1/UI/ [Subject to change]
2. If you don't already have one, create a folder for your work in the System/Templates/Sandbox/ folder. Name it "[Your Name] Templates".
    a. Contact [training@crownpeak.com] if you need an account with access to this sandbox.
3. Make a copy of either the template you created in training or the /System/Templates/TDW/TDW Template Example.
    a. You can copy/paste in the CDC or in the CMS to make a copy of the template
4. Paste the Template into your /System/Templates/Sandbox/{template folder}.
5. Rename it "Widget Container Template".

↗ Lesson 2: Assign the template to a test asset

1. In the CMS, navigate to your /Sandbox Widgets/{Your Name} folder (NOT in the System folder. In the site folders)
    a. If it doesn't exist, create this new folder.
2. Use **New > File** to create a new asset. Name it "Widget Container Page" and assign it to your new template.
3. Assign Basic workflow to your asset.
4. Edit the test file and enter some content. For example, insert a contact name or two in the sidebar tab.
5. Now preview the template. Note the sidebar area with the contact numbers.
6. Note that this is the code we are going to be moving into a widget.

↗ Lesson 3: Identify the code to be widgetized

1. In the CDC, open your Container template in /System/Templates/Sandbox/, and open the input.aspx and the output.aspx in Visual Studio.
2. In Input.aspx, find the code that creates the contact panel:

```
Input.ShowMessage("Enter Contact Names (in right rail area).");
 // List Panel
 while (Input.NextPanel("panel_counter", displayName:"Contact
Information"))
 {
Input.ShowTextBox("Name", "name");
Input.ShowTextBox("Telephone", "telephone");
 }
```

3. In Output.aspx, find the code that outputs the panel:

```
<div id="sidebar">
<%
List<PanelEntry> sectionItems = asset.GetPanels("panel_counter");
foreach (PanelEntry sectionItem in sectionItems)
{
%>
<div class="box">
<div class="box-content">
<%
Out.WriteLine("<h1>{0}</h1>", sectionItem["name"]);
Out.WriteLine("<h1>{0}</h1>", sectionItem["telephone"]);
%>
</div>
</div>
<div class="box_divider"></div>
<%  } %>
</div><!--/ sidebar -->
```

↗ Lesson 4: Create a new Widget Template

1. In the CDC, create a new Template C# in /System/Templates/Sandbox/[Your Template Folder] and call it Contact Widget.
2. Open the input.aspx and output.aspx for the widget template in Visual Studio.
3. Add the following controls to control the Widget's label and an optional field to control the Widget's Title.

```
Input.ShowTextBox("Widget Label", "page_label");
Input.ShowTextBox("Widget Title", "page_title");
```

4. Create a post_input.aspx for the Widget template and add the following code:

```
<%  TDW.SetAssetLabel("page_label", context, asset); %>
```

5. You may remember that this function keeps the widget name in sync with the page label field.

↗ Lesson 5: Move the I/O fields to the new template

1. In the input.aspx, complete the following
    a. Select the following code from the **container** template

```
Input.ShowMessage("Enter Contact Names (in right rail area).");
// List Panel
while (Input.NextPanel("panel_counter", displayName:"Contact
Information"))
   {
      Input.ShowTextBox("Name", "name");
      Input.ShowTextBox("Telephone", "telephone");
   }
```

    b. Cut these fields from the container's input.aspx and paste them to the **widget's input.aspx**, after `Input.ShowTextBox("Widget Title", "page_title");`
    c. Now, replace the list panel in the **container** page with the following document selection panel:

```
Input.ShowMessage("Select and sort widget assets.");
      ShowAcquireParams sParams = new ShowAcquireParams();
      sParams.ShowUpload = false;
      sParams.DefaultFolder = "/" + asset.Parent.AssetPath +
"/Widgets/";
      while (Input.NextPanel("widget_counter":
displayName:"Widgets"))
      {
            Input.ShowAcquireDocument("Select Widget",
"widget_document", sParams);
      }
```

    d. This panel will give you the flexibility to select and reorder multiple widgets. It adds two parameters to the document selection control: setting the default folder to a "Widgets" folder in the Container asset's folder (easily customized), and disallowing uploading external files (this widget template won't work with uploaded PDFs or Images…)

2. In the output.aspx, create output code for the widget and include API methods to display the widget output in the container page
    a. First select the following from the **container's** output code:

```
<%
      List<PanelEntry> sectionItems =
asset.GetPanels("panel_counter");
      foreach (PanelEntry sectionItem in sectionItems){
```

```
            %>
                        <div class="box">
                            <div class="box-content">
                                <%  Out.WriteLine("<h1>{0}</h1>",
sectionItem["name"]);
                        Out.WriteLine("<h1>{0}</h1>",
sectionItem["telephone"]);
                        %>
                    </div>
            </div>
                        <div class="box_divider"></div>
                        <% } %>
```

b. Paste it into the **widget** template's output.aspx.
    i. Note that this will not publish a full html page. It has no nav wrap, and will only create a section of code, which is typical for a "widget".
c. Now add the following code inside of the "sidebar" `<div>` in the **container** page:

```
<%    List<PanelEntry> widgetItems =
asset.GetPanels("widget_counter");
      foreach (PanelEntry widgetItem in widgetItems)
            {
                  Asset widget =
Asset.Load(widgetItem.Raw["widget_document"]);
                  if (widget.IsLoaded)
                        Out.WriteLine(widget.Show());
            } %>
```

↗ Lesson 6: Publishing Configuration

1. Next consider publishing options. The simplest option is illustrated above, the widget code will be rendered as part of the container page code.
2. An additional step to keep this configuration of widget from publishing a separate file is to create a filename.aspx in the widget template, then add the following line of code to it:
   ```
   context.PublishPath = "";
   ```
3. If you have a widget that will be updated regularly and you need to be able to publish it separately to it's container pages, you might consider some advanced options for publishing the code, discussed below.

↗ Lesson 7: Create Model and your first widget asset

1. Next create a subfolder to become your Widget Model
2. Navigate to /System/Models/Sandbox/[Your Models Folder]/ and create a new folder called [Widgets Model]
3. In this new folder, create a new file, call it “Contact Widget”, assign it to your widget template, then assign it to the basic workflow and change the asset properties to “inherit from parent”.
4. Navigate back to the folder that contains your container asset, and create a new subfolder called “Widgets”. Select this folder and set the **Properties > Model** to your new Widgets Model folder.
5. Navigate inside the new widgets folder, verify that **New > Contact Widget** is an option, and create a new contact widget asset. Enter a few contact numbers and save.
6. Now navigate back to your container page, edit it and click the “form” view. In the Widgets tab, Browse to and select your new widget.
7. Preview to check that the widget displays.

↗ Lesson 8: Publishing Order

1. With both your widget and container page having workflow, both assets need to be published in the same state in order for the widget to appear correctly on the published page.
2. As a general practice, it's best to begin publishing with digital files, then script assets, then move to widgets, then container pages, then index pages.
3. In this case we're only working with two assets, so publish the widget asset to stage first, then publish the container page to stage.
4. Now verify the page and widget display correctly on the published server.

ADVANCED WIDGET DISCUSSIONS

Below are some advanced discussion topics for widgets. If you need further support or clarification, please feel free to contact CrownPeak training (training@crownpeak.com) or support (support@crownpeak.com), or reach out to our user community on connect.crownpeak.com.

↗ How to handle multiple widget types?

There is often a need for different types of widgets that can be added to pages interchangeably (e.g., a wysiwyg box, a related links callout, a localized search or browse box, etc.). There are a few options for configuring these options in CrownPeak.

One option is to create multiple widget templates to handle the different types (i.e., add separate templates). In this case, you could create copies of your widget template and customize them as needed. Then add the additional widget types to your model (i.e., then they'd show up in your **New** menu as additional file options, so that in addition to **New > Contact Widget**, your users will have options for **New > WYSIWYG Widget**, **New > Relate Links Widget**, etc. This might be the best option if there are only a few widget types that you need to implement and the types are distinct (An end user is unlikely to confuse them).

A second option is to expand your widget template with types/cases for handling different subtypes. This can be done by adding an `Input.StartDropDownContainer()` in the input defining the different widget options, and then adding matching `switch() {case}` code in the output. This option might be the best to implement if there are a large number of variations, especially if there are common fields you would like to reuse and then you can set up the field groups that are different in the `DropDownContainer` while keeping the rest of the input consistent.

↗ Improving widgets in preview

You may have noticed that previewing your widget doesn't show styled content. If needed, a separate preview template file can be added that will allow the end user to view the widget with the applicable styles. You might consider applying your navwrap to the preview and load the existing widget output code with `Out.WriteLine(asset.Show("output"));` You may need to add some containing `<div>` tags to the preview template with appropriate class/id attributes to make the widget display correctly depending on your site's HTML/CSS.

If the nav wrap is elaborate or doesn't display correctly, or you have a large number of widgets you'd like to be able to preview, you might consider creating a copy of the navwrap (a "preview

wrap") just for previewing. This might increase code maintenance burden, however. If this is a preferred option, you may wish to set up the styles and scripts of your nav wrap either in a function or in a separate asset that you could include in each wrap without having to maintain two sets of code.

↗ Publishing as a PHP Include, SSI (Server-side Include) or .Net User Control

If you have widgets that are regularly updated or that are included on a number of pages, you can configure them to publish separately and include their code as a server-side include, PHP include or .Net user control depending on your server platform.
Publishing as a .net user control is also possible, but requires registering the control on the "containing" page.

Please contact CrownPeak support to help select and set up an appropriate publishing option if you wish to publish your widgets as includes..