CrownPeak™

TECHNICAL DOCUMENTATION

C# API
VBSCRIPT API TRANSITION GUIDE
VERSION 1.0

September 2011

TABLE OF CONTENTS

VBSCRIPT AND C#

One of the major differences between the old API and the new one is that the language has changed from VBScript to C#.
C# is an object-oriented programming language. For those unfamiliar with the concept, please review the following links.

1. Object Oriented Programming Concepts: http://download.oracle.com/javase/tutorial/java/concepts/
2. Structure of a C# program: http://msdn.microsoft.com/en-us/library/w2a9a9s3.aspx
3. Visual C# Introduction (Video): http://www.microsoft.com/events/series/msdnvs2010.aspx?tab=Webcasts&seriesid=156&webcastid=17803

C#'s syntax is very similar to Java, so developers who have experience with Java should feel comfortable moving to C#.

The links below are general resources on transitioning from VBScript to C#

4. Programming Concepts Compared in Different Languages with Code Examples: http://msdn.microsoft.com/en-us/library/we7h0cz1(v=VS.80).aspx
5. Language Equivalents: http://msdn.microsoft.com/en-us/library/czz35az4(en-US,VS.80).aspx
6. Migrates from VBScript Functions to C#: http://www.netcoole.com/asp2aspx/vbhtml/csfuncs.htm
7. From VB.NET to C# and Back Again (From VB to C# case study): http://www.4guysfromrolla.com/webtech/012702-1.shtml
8. Converting Mike Shaffer's VBScript RC4 Encryption Code to C# (From VB to C# case study): http://www.4guysfromrolla.com/articles/091802-1.aspx

Below are reference guides for C#.

1. The C# language: http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx
2. C# reference: http://msdn.microsoft.com/en-us/library/618ayhy6.aspx

Access to the C# API documentation is available at http://help.crownpeak.com/cmsapi

KEY DIFFERENCES BETWEEN VBSCRIPT API AND C# API

So, there are some key syntactical differences that developers with experience with the old API should note

- C# is case sensitive
- Code statements are generally terminated with semicolons.
- Code blocks are enclosed with braces { }
- Single-line comments follow two slashes // like this, multi-line comments may be nested between slashes and asterisks /*like this*/

And setting aside syntactical differences between the old VBScript API and new C# API, the new one follows a more modern design paradigm.

- Variables are strongly typed
- Variable casting (converting variable from one type to another) requires explicit action
- Most importantly, the old API, based on VBScript, used pre-defined and user-defined functions and subroutines. The new API, is based on C#, an object-oriented language, uses classes with properties and methods.

CODE REUSE

Rather than using include files stored in the template folders and accessed using `system.include`, the C# API makes it possible to create functions and share them through a custom library. Functions and objects that are defined in the custom library are available to all template files. We strongly recommend that any code that is intended to be used in a variety of templates be kept here to maximize code reuse. Library files are stored in your CMS under **/System/Library/**. The **Custom.cs** file is available by default in each CMS instance. You can add code for reuse to this file or create additional files to organize your functions and objects.

SETS VS OBJECTS

In order to get information about an asset under the VBScript API, the developer needed to make a function call to fetch the content which was then returned as a dictionary that would typically be assigned to a set.
The C# API now represents an asset's content with an Asset object.

CUSTOM FIELDS AND CMS META INFORMATION

Under the VBScript API, the dictionary which held the content contained a mixture of the custom fields defined in the templates and CMS meta fields. These meta fields were prepended with `_cms` such as `_cmsLabel`. Under the C# API, the `_cms` prefix has been dropped and these fields have been converted into object properties.

Below is an example of how to access the label, id, and path asset object properties.

VBScript (Old)

```
set fields = asset.getContent("/Site/Home Page");

response.write("Label: " & fields.item("_cmsLabel"))
response.write("Id: " & fields.item("_cmsId"))
response.write("AssetPath: " & fields.item("_cmsFolder"))
```

C# (New)

```
Asset homePage = Asset.Load("/Site/Home Page");

if (homePage.IsLoaded)
{
Out.WriteLine("Label: " + homePage.Label);
Out.WriteLine("Id: " + homePage.Id.ToString());
Out.WriteLine("AssetPath: " + homePage.AssetPath.ToString());
}
```

In the example above, the static function, `Asset.Load()`,loads the asset located in the cms at "/Site/Home Page" and is stored in `homePage`. The `Label`, `Id`, and `AssetPath` are accessed as properties of the asset.

Note that these properties are not always stored as strings and in C# need to be cast as strings – `homepage.Id.ToString()` - to display in HTML output or to use in string operations.

Additionally, we've implemented a new best practice, where we check that the asset object is loaded with the ".IsLoaded" property before trying to display the fields. This is necessary when loading content from an external asset, not the current one.

To access the content stored within an asset, simply load the asset into an `Asset` object and then use its indexer to access the appropriate field. The example below shows how to access custom fields through the new and old API.

VBScript (Old)

```
set fields = asset.getContent("/Site/Home Page")

response.write("Title: " & fields.item("title"))
response.write("Short Title: " & fields.item("short_title"))
response.write("Description: " & fields.item("description"))
```

C# (New)

```
Asset homePage = Asset.Load("/Site/Home Page");

if (homePage.IsLoaded)
{
Out.WriteLine("Title: " + homePage["title"]);
Out.WriteLine("Short Title: " + homepage["short_title"]);
Out.WriteLine("Description: " + homePage["description"]);
}
```

In the example above, we are loading the asset and then accessing custom fields via the indexer. We are then accessing the value of description by using `homePage`'s indexer.

Note that all content fields are stored as strings and in C# must be cast to other datatypes (numeric, date, etc.) to perform any non-string calculations/operations, and possibly back to strings again for HTML output.

Here is an example where we retrieve a date string, convert it to a date, and re-output it as a formatted date string:

```
Asset article = Asset.Load("/Site/Articles/news20110101");

if (article.IsLoaded)
{
Out.WriteLine(Convert.ToDateTime(article["posting_date"]).ToString("d"));
}
```

The new API uses standard C# date functions for formatting date strings. Reference:
http://msdn.microsoft.com/en-us/library/az4se3k1.aspx
http://msdn.microsoft.com/en-us/library/8kb3ddd4.aspx

CONTEXT OBJECT

The Context object makes data about the current asset available through its properties. There are different context objects available in each template file. For example, in output.aspx, a CMS developer has access to the OutputContext object, that includes properties such as `IsPublishing` and `LayoutName`. In the VBScript, this data was available as a part of the content object from field names beginning the `_cms`.

The following example illustrates the difference between accessing this data in VBScript and C#. In a VBScript filename.asp, you would get the name output file being used for the publishing with

```
content.item(“_cmsLayout”)
```

This would return the name of the template file. However, in a C#, you would get this filename using the OutputContext object like this:

```
context.LayoutName;
```

The following example shows that the way we access content can be different depending on the template file. In a VBScript input.asp and post_input.asp file, we could access a data field called "data_field" as

```
content.item("data_field")
```

However, in a C# post_input.aspx file, we would refer to "data field" as being a part of the PostInputContext’s InputForm property, like so:

```
context.InputForm["data_field"];
```

In the input.aspx file, this field is managed by the Input class and method used to create the field.

CONCATENTATED STRINGS AND STRINGBUILDER

The above example creates output strings with the concatenation operator (+ in C#). While this is fully supported in the C# API, it is often considerably more efficient to use a C# class called `StringBuilder` to create longer strings and complex chunks of output HTML. Using stringbuilder in iterative output templates can both simplify code appearance and decrease publishing times. General information about the `StringBuilder` class is here: http://msdn.microsoft.com/en-us/library/2839d5h5%28v=vs.71%29.aspx

OUTPUTTING INDIVIDUAL FORMATTED STRINGS

C# (Formatted Strings)

```
Asset homePage = Asset.Load("/Site/Home Page");

if (homePage.IsLoaded)
{
Out.WriteLine("Title: {0}", homePage["title"]);
Out.WriteLine("Short Title: {0}", homePage["short_title"]);
Out.WriteLine("Description: {0}", homePage["description"]);
}

// here is an example of a formatted string with multiple parameters

Out.WriteLine("Name: {0} Phone Number: {1}", asset["name"],
asset["number"]);
```

BUILDING HTML WITH STRINGBUILDER

If you are building a chunk of code for output, it is a best practice to build the code in a stringbuilder object.
The following is an example of a function for building the meta portion of a page.
Note that the stringbuilder object must be cast as a string for final output.
Also note that we must append linefeeds to the stringbuider object.

C# (Stringbuilder)

```
public static string MetaStringBuilder(Asset asset, Asset globalConfigAsset)
{
        StringBuilder metaOutput = new StringBuilder("");
        String metaTitle;
        String metaDescription;
        String metaKeywords;

if (!String.IsNullOrWhiteSpace(asset["meta_title"])
{
        metaTitle = asset["meta_title"];
} else {
        metaTitle = globalConfigAsset["meta_title"];
}

if (!String.IsNullOrWhiteSpace(asset["meta_desc"])
{
        metaDescription = asset["meta_desc"];
} else {
        metaDescription = globalConfigAsset["meta_desc"];
}

if (!String.IsNullOrWhiteSpace(asset["meta_keywords"])
{
        metaKeywords = asset["meta_keywords"];
} else {
        metaKeywords = globalConfigAsset["meta_keywords"];
}

metaOutput.AppendFormat("<meta name=\"{0}\" content=\"{1}\" />", "title",
metaTitle);
metaOutput.AppendLine();

metaOutput.AppendFormat("<meta name=\"{0}\" content=\"{1}\" />",
"description", metaDescription);
metaOutput.AppendLine();

metaOutput.AppendFormat("<meta name=\"{0}\" content=\"{1}\" />", "keywords",
metaKeywords);
metaOutput.AppendLine();

return metaOutput.ToString();
}
```

STATIC AND NONSTATIC FUNCTIONS

Static functions are functions that are accessible directly from the class, while non-static functions are accessible from an instance of the class. Non-static functions are often used for implementation specific to an instance of a class, while static functions are often used for more generic functions, such as adding two parameters together or helper functions that wrap data in markup (html, xml, etc.). consider the example code below:

Example of Static and Non-Static Functions (From Custom.cs)

```
public class Custom
{
  public static string ReverseString(string reverseThis)
  {
    //code to reverse the string
  }

  public void string ReverseString(string reverseThis)
  {
    //code to reverse the string
  }
}
```

Example of Output.aspx using functions

```
<%
   string original = "This string will be reversed";

   //Calling the static version of ReverseString
   Out.Write(Custom.ReverseString(original));

   //Calling the non-static version of ReverseString
   Custom customInstance = new Custom();
   Out.Write(customInstance.ReverseString(original));
%>

Output:

desrever eb lliw gnirts siht
desrever eb lliw gnirts siht
```

FUNCTIONS

FUNCTION MAPPING

Class	VBScript (Old)	C# (New)
Asset	setContent	SaveContent
	getLink	GetLink
	getFileList	GetFileList
	getFolderList	GetFolderList
	getFilterList	GetFilterList
	create	CreateNewAsset
	rename	Rename
	move	Move
	addDependency	AddDependency
	getBinaryLink	GetBinaryLink
	setSchedule	SetSchedule
	clearSchedule	ClearSchedule
	route	Route
	getDateOverlap	GetDateOverlap
	setLabel	Rename
	deleteContentField	DeleteContentField
	createListFromFolder	GetPanelsFromFolder
Image	getHeight	Height (this is a property)
	getWidth	Width (this is a property)
	thumbnail	CreateThumbnail
Input	checkBox	ShowCheckBox
	showAcquireDocument	ShowAcquireDocument
	startTabbedPanel	StartTabbedPanel
	nextTabbedPanel	NextTabbedPanel
	endTabbedPanel	EndTabbedPanel
	showAcquireImage	ShowAcquireImage
	showSelectDate	ShowSelectDate
	showSelectStr	ShowSelectStr
	showTextArea	ShowWYSIWYG

	showSelectDocument	ShowSelectDocument
	startExpandPanel	StartExpandPanel
	endExpandPanel	EndExpandPanel
	showSelectColor	ShowSelectColor
Sys	setDependency	SetDependency
	startCapture	StartCapture
	stopCapture	StopCapture
User	inGroup	IsInGroup
	getUser	GetUser
Util	filterText	FilterText
	stripHTML	StripHtml
	convertTextToHtml	ConvertTextToHtml
	escapeItem	EscapeItem
	crop	Crop
	titleCase	StartCase
	filterFilename	FilterFilename
	showEditButton	ShowEditButton
	getHttp	GetHttp

MOVED FUNCTIONS

While the new API shares a lot of similar functions with the old API, a few functions were moved to more appropriate places. The chart below shows the mapping between the old and the new.

VBScript Class	VBScript Name	C# Class	C# Name
response	write	Out	Write
system	Wrap	Out	Wrap
content	isPublishing	context	IsPublishing (this is a property)
content	isNew	context	IsNew
content	escapeItem	Util	EscapeItem
asset	createListFromCSV	Util	CreateListFromCsv
system	setDependency	context	IsGeneratingDependencies

content	add "_saveedit", 1	context	ShowSaveEditButton
content	add "_savenew", 1	context	ShowSaveNewButton
	N/A – controlled in ACL	context	ShowPublishButton
	N/A – controlled with availability of preview or output template?	context	ShowSavePreview

REMOVED FUNCTIONS

Below is a list of functions that were NOT ported to the new API. Many of the vbscript functions that were not ported over were removed.

Name	Notes
asset.getId	Replaced by a property: Id
asset.delParam	Removed since we now pass the params object to the function that needs it.
asset.getParam	Removed since we now pass the params object to the function that needs it.
asset.setParam	Removed since we now pass the params object to the function that needs it.
asset.getContent	Removed since the content is immediately available after loading the asset.
asset.getLabel	Replaced by a property: Label
clock.formatDate	Replaced by the DateTime Class which is part of C#
clock.getChicagoName	Replaced by the DateTime Class which is part of C#
clock.getDate	Replaced by the DateTime Class which is part of C#
clock.getDateTime	Replaced by the DateTime Class which is part of C#
clock.getMonthName	Replaced by the DateTime Class which is part of C#
clock.getTime	Replaced by the DateTime Class which is part of C#
clock.getWeekdayName	Replaced by the DateTime Class which is part of C#
clock.isFutureDate	Replaced by the DateTime Class which is part of C#
clock.isValidDate	Replaced by the DateTime Class which is part of C#
content.add	Replaced by an Indexer: asset["title"] = "This is my title";
content.item	Replaced by an Indexer: asset["title"];
content.itemAt	The way we access lists has been changed. As a result this function has been removed.
content.defaultItem	Propose that we move this convience method to the CMSCustomLibrary
debug.log	Don't need this function because Debug.Write does the exact same thing
listHelper.item	Replaced by C# List
listHelper.itemAt	Replaced by C# List
listHelper.itemIndex	
listHelper.nextEntry	
listHelper.nextPanel	
listHelper.size	Replaced by C# List
response.write	Replaced with Out.Write
content.isTrue	Probably don't need this anymore. We can leverage C#

REGARDING LIST PANELS

Under the VBScript implementation of the CMS, accessing lists was complicated. In order to truly understand how to use a list, the developer had to have a deep understanding of the implementation details associated with the list, particularly for multidimensional lists. The method for creating the list was always the same, but displaying the list in the input.asp required a different method than output.asp, `nextPanel()` and `nextEntry()` respectively. However, it did not restrict the usage where appropriate. As a result, the developer was often left surprised that input controls were showing up in the output.asp.
Under the C# implementation, implementation details have been abstracted away and the methods for displaying lists in the input and output have been moved to the Input class and Asset class respectively. This ensures that the panels are displayed correctly no matter where they are accessed within the CMS.

INPUT

Consider the code below both snippets create a list of fields called short_title.

VBScript (Old)

```
  <% set list = content.createList("counter")
     do while list.nextPanel() %>
  <input type="hidden" name="counter" value="<%= list.itemIndex %>" />
  <table width="100%" class="tabletext">
    <tr>
      <td>Title</td>
      <td>
        <input name="title" size="50" value="<%=
list.escapeItem("short_title") %>" />
      </td>
    </tr>
  </table>
  <% loop %>
```

C# (New)

```
    while (Input.NextPanel("counter"))
    {
        Input.ShowTextBox("Short Title", "short_title");
    }
```

Note that the C# code is much shorter than the VBScript code. Since the new API renders the HTML for the input screen rather than leaving it up to the developer, the implementation details are now also being abstracted. In the VBScript example, the hidden field counter was an implementation detail that ensured that a list that contained empty values in the middle would not end inexplicably when being rendered in the output. The new API automatically adds the necessary implementation code while it renders the page.

The important thing to be aware of is that `while(Input.NextPanel("counter"))` is roughly the equivalent to the first three lines of the vbscript example. To add more controls to the list panel, it is as simple as adding another control within the while loop's code block. . The following example also demonstrates setting upload parameters for an image field.

```
ShowAcquireParams imageParams = new ShowAcquireParams();
    imageParams.Extensions = Util.MakeList("png", "jpeg", "jpg", "gif");
    imageParams.ShowBrowse = true;
    imageParams.ShowUpload = true;

    while (Input.NextPanel("counter"))
    {
        Input.ShowTextBox("Short Title", "short_title");
        Input.ShowAcquireImage("Image", "image", imageParams);
        Input.ShowTextBox("Caption", "caption");
    }
```

OUTPUT

To output the above loop, we retrieve the panels in a list and the iteratively return the contents. Here we also demonstrate an example of loading an image object and returning the height and width parameters.

```
List<PanelEntry> contentPanels = asset.GetPanels("counter");

foreach (PanelEntry contentPanel in contentPanels)
    {
        Img image = Img.Load(contentPanel["image"]);

        Out.WriteLine("<h3>{0}</h3>", contentPanel["short_title"]);
        Out.WriteLine("<img src=\"{0}\" alt=\"{1}\" height=\"{2}\"
width=\"{3}\" />", contentPanel["image"], contentPanel["short_title"],
image.Height, image.Width);
        Out.WriteLine("<p>{0}</p>", contentPanel["caption"]);
    }
```

TIPS FOR PORTING VBSCRIPT TEMPLATES TO C#

WRAP THE VBSCRIPT CODE WITH /* */

1. Paste the VBScript code into the C# template file and wrap with /* asp code */. This will allow you to save the template file. As you port portions of the code, delete them from inside the commented area.

DEFINING METHODS IN A C# TEMPLATE FILE

```
<script runat="server" data-cpcode="true">
   string SomeMethod(Asset Asset)
   {
      return "some string";
   }
</script>
```

3. This is a good way to test your Custom Libary methods before placing them there. Since you can't debug Custom Library classes yet.
4. Additionally, template-specific methods can be defined in these code blocks and kept separate from the page code to make code more readable.

Note: if you want to define a server-side C#.NET function in a published page, remove the data-cpcode="true" attribute from the script tag.

CREATING C# TEMPLATES

Always use "New Template C#" to create a folder to store C# templates, do not put C# templates into folders created by other means.
Do not mix C# template files with ASP template files in the same template folder.

DESKTOP CONNECTION/VISUAL STUDIO 2010 EXPRESS SETUP

1. Uninstall Desktop Connection if you are not on the Beta channel.
2. Install Desktop Connection from the Beta channel: http://desktop.crownpeak.com/dc/standalone/beta/publish.htm
3. Install Visual Studio 2010 Express (Visual Web Developer 2010 Express http://www.microsoft.com/express/Downloads/#2010-Visual-Web-Developer
4. Configure Desktop Connection:
5. Set Editor Path to:
   C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\VWDExpress.exe
   or
   C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\VWDExpress.exe
   based on your OS version.
   Devenv.exe replaces VWDExpress.exe when using Visual Studio 2010 Professional.
6. Restart Desktop Connection
7. Right click on the Root folder ("/") and click on Install Visual Studio 2010 Development Environment. This is required for debugging.
8. Edit a C# template file in Desktop Connection, this should open up an instance of Visual Studio.
9. Right click on the CMS project and select Build, this is needed at least once to build up the Intellisense files.

ENABLING CODE SNIPPETS IN VISUAL STUDIO

1. In Visual Studio, select Tools > Code Snippet Manager from the top menu
2. Set language to C#
3. Choose add, then navigate to and select C:\Documents and Settings\All Users\Application Data\CrownPeak\VSCodeSnippets\CrownPeak
4. Click OK. In template file, type cp. You will see Intellisense documentation for the available snippets. Click tab twice to insert the full snippet.

Code snippets include:

1. Variables are highlighted so it is clear what can be edited.
2. Object names are managed. If object name is changed in one location, it is updated throughout snippet.

For a video on this Visual Studio feature, go to http://msdn.microsoft.com/en-us/vstudio/ee625748.aspx

HOW TO DEBUG A C# TEMPLATE

1. Click margin to set breakpoint.
2. Click F5 to trigger debugging. The first time, it will ask you to create a web.config file.
3. Open the Watch Panel, using Debug > Windows > Watch. Type 'asset' in the available name field. This will populate with information on the page you just created. You will be able to monitor changes in the watch panel as you modify the page in the CMS.

WORKAROUNDS

1. CS files and Intellisense
    a. Use Standalone CDC
    b. In Solution Explorer, select 'CMS'. Right click and choose Add > Existing Item
    c. Navigate to the local version of the .cs file created by CDC. Should be available in a path like - C:\Documents and Settings\All Users\Application Data\CrownPeak\cpPrefs\a25\MasterTrainingInstance\System\Library
    d. Select the file. Intellisense should be available. If not Clean and Build.

KNOWN ISSUES

1. Workflow email template attachments are not supported
2. Workflow email content items were visible in workflow exec file template using old API, not with the new one.
3. Two closing braces same line can cause format string exception
4. Asset.route: New shortcut will always refer to draft. Old API did not, but the UI did.
5. Some server-side C# code must be escaped to prevent the CMS from compiling it as CMS API code.
6. Some invalid context-sensitive code checking in Visual Studio 2010
7. API Intellisense not available in .cs Library files