Versioning Library Classes and Methods

Version 1.0

Document History

Author/Editor	Date	Reason for Change	Version
CrownPeak	05/2013	Original Version	1.0

Overview

The CrownPeak CMS C# API uses a custom library where custom classes can be added for use by CMS templates. Only one library is supported per instance. Updated methods can be tested safely without impacting production pages until those changes are verified and approved.

There are a few different ways to version library classes and class methods. Developers should determine which approach best suits their needs.

Assumptions

This document assumes that the developer is using alternate outputs for stage and live (and potentially other states if applicable) to control template changes. You can review *Configuring CrownPeak for Multiple Projects and Developers Best Practices* on Connect for information about maintaining multiple outputs of templates.

Versioning Methods

Versioning methods is the simplest way to control the versions and methods in use.

Example 1

Example one contains a simple class sample with a method sample V1. To change the behavior of this method create a method numerating the version. To do this, copy the method, paste it below, and rename the method. See Example 2.

```
namespace CrownPeak.CMSAPI.CustomLibrary
{
  public class Sample
  {
      public static string Sample()
      {
          string results = "v1";
          return results;
      }

  }
}
```

Example 2

To control deployment of the change in the *output-live.aspx* retain the old version and **Live** is unchanged.

```
Out.WriteLine(Sample.Sample());
```

Update *output-stage.aspx* to use the updated version.

```
Out.WriteLine(Sample.SampleV2());
```

Once the changes are verified and approved, promote *output-stage.aspx* to *output-live.aspx*, to promote the new method to **Live**. In the event the change needs to be rolled back, you can revert to the previous version of the method by reverting *output-live.aspx* to the previous version. You can also change the call line of code.

```
namespace CrownPeak.CMSAPI.CustomLibrary
{
  public class Sample
  {
      public static string Sample ()
      {
          string results = "v1";
          return results;
      }

      public static string SampleV2()
      {
          string results = "v2";
          return results;
      }
  }
}
```

Versioning Classes

You can control versioning by creating alternate class files. In some cases the entire collection of methods may need to be changed. In this scenario clone *Sample.cs* creating *SampleV2.cs*. To create the new cs file you must immediately rename the class declaration to the new class. The CMS cloning process does not rename the class when it is cloned. If the class is not renamed the custom library will not compile.

Example

```
namespace CrownPeak.CMSAPI.CustomLibrary
{
  public class Sample
  {
      public static string Sample()
      {
          string results = "v1";
          return results;
      }

  }
}

Example 4:

namespace CrownPeak.CMSAPI.CustomLibrary
{
  public class SampleV2
  {
      public static string Sample()
      {
          string results = "v2";
          return results;
      }

  }
}
```

Now the new class has been created change the same line in *output-stage.aspx* to call the new class from:

```
Out.WriteLine(Sample.Sample());
```

To

```
Out.WriteLine(SampleV2.Sample());
```

Once approved *output-stage.aspx* can be promoted to **Live**. If a rollback is necessary, the original *output-live.aspx* can be reverted.

Notes

- If a class name or method name is changed other templates may need to be updated in addition to the template that is currently being tested. Remember to locate all templates that use that method. The CDC has a search function that works well for this task.

- In some cases a method may add new parameters. When updating the templates calling this method, be sure to update the arguments accordingly. One method to minimize the impact is to make the new parameters optional.

- An alternative to using the alternate output templates is to wrap the calling statements in a conditional statement based on *context.PublishingPage.PackageName* (where it is published) or *context.IsPublishing* (preview or published version).