CMS and HTML Requirements and Recommendations

Table of Contents

Introduction

This document provides a set of requirements and recommendations for developers on providing HTML, JavaScript, CSS, and other client-side (and on occasion, server-side) code to CrownPeak for integration into templates. There are 4 sections to this document:

- Planning recommendations and best practices. There are some front-end coding practices that will make your code more easily portable to templates. All HTML, CSS, JavaScript can be supported, but these are the issues to consider during planning that will reduce the need for refactoring.
- Required Deliverables. Items in this section are required elements of the code to be delivered. The Implementation portion of your CMS project cannot begin until these are satisfied.
- Recommendations. This section contains items that we recommend be a part of your code delivery, but are not requirements for your project to begin or to be completed.
- Other Components. These items are ones that would be “nice to have,” but are mostly suggestions to the developers.

This is a general document and intended to apply to a broad range of projects. Please feel free to speak with your Account Representative with any questions you might have, as portions of the document may not apply to you.

Planning Recommendations and Best Practices

When designing and developing front end code (HTML, CSS, and JavaScript) for integration with CrownPeak, consider these best practices

QIQO (Quality in, Quality out)

CrownPeak will create an implementation to support the site EXACTLY. That means we take the client-side (and if needed, server-side) code and use it implement templates. We will not update or change features or functionality unless explicitly engaged to do so. Therefore, all the code delivered should do EXACTLY what the site will, ultimately. It should include all the styles, all the functionality, all the features that need to be available on the site.

Holistic Code Creation

CrownPeak views all pages on a site as part of a whole. They often share styles, layouts, features and formats. When configuring CrownPeak, we analyze the front-end code for these shared elements - it drives template implementation and configuration decisions.

Some best practices for holistic code creation:

- Create a framework for page development. Each page in the site should not be created in a vacuum. Each page should build on existing styles and scripts. Our recommendation is to start development with a page type that can be used as a framework to develop the others.
- Take the same approach with CSS. Don't create separate CSS for each page and don't create CSS without understanding the existing styles. As much as possible, reuse and build upon the existing styles.
- Create HTML for every page type that will need to be managed in CrownPeak. Even if the page uses elements and formats available on other pages, having an example will make it clear what needs to be created by CrownPeak and give you an opportunity to validate the layout.

For responsive sites, some best practice to consider:

- Create breakpoints for standard device sizes
    - 320X480
    - 600X800
    - 1024X768
    - Desktop
- For each breakpoint, create CSS to support the portrait and landscape versions
- Define CSS using @media attribute. Best practices available at *http://css-tricks.com/css-media-queries/*
- Use % and em dashes to specify dimensions instead of pixels

Reusing and Maintaining Code

Consider what each page type will do on the site and the content it will support.

Will users need the option enter text, images or videos into the same area? Does the HTML support the different requirements for these content types?

Will the pages be used to support sites in multiple languages?

- Make sure all text, like navigation and headers, is HTML, not graphic, so they can be easily updated
- Does the content area provide enough space for all languages?

Style names and ids, and JavaScript variables should be labeled so other developers quickly understand their role. For example, a style for a two column layout should be called “two_col”, not “style_one”

JavaScript functions should be developed so they can be reused. For example, instead of hard-coding specific ids, paths, or names set these up as parameters in the function so the JavaScript can be repurposed for other pages or templates.

Testing and Updates

Before delivering HTML to CrownPeak, determine the browser and devices your site should support. Then, validate, test, and repair, if necessary. Remember - Quality in, Quality out!

If you must change your code, edit the existing code instead of starting from scratch

Document and send a list of changes to CrownPeak. Updates to front-end code usually require templates and configurations are re-factored.

Required Deliverables

The following items are required in order to begin the Implementation portion of your CMS project.

Client Approval

All code/design/images/layout must be approved by the client prior to delivery to CrownPeak.

All Elements Must Be Functional

All elements to be implemented in the CMS or that contain CMS managed content must be functional. For example, photo slideshows, printable versions, and email this content forms, etc.

Browser Compatibility

Delivered code must be fully tested on all target browsers and operating systems specified by the client. Untested, unapproved code will not be accepted.

Validation

Delivered code should validate to the DOCTYPE standard specified, usually XHTML 1.0 Transitional or XHTML 1.0 Strict. Acceptable validators include W3C (http://validator.w3.org).

Plan for Changing Content

Delivered code should be built to plan for adjustable content: images that may or may not be provided, text that varies greatly in length, etc. We have seen issues where code has been built using the same FPO (for placement only) content, and later does not scale properly for pages that have widely differing lengths. Required items should be noted in documentation accompanying the code delivery.

Slice

The extranet should include a "slice" implementation of the site. This would be an implementation of a section of the site illustrating the functional and navigational path of one portion or "slice" of the content. This is invaluable in ensuring that navigational and functional issues have been identified and resolved prior to CMS implementation. The slice does not have to be separate from the code delivery - it should just be a portion of the code delivered.

Extranet Access

Delivered code must be posted in a functional, browsable format on an extranet for the duration of the project. This gives us a point of reference as we implement the code into the CMS templates. Updates to the code during the course of the project must be posted to the extranet to ensure an accurate reflection of the look and feel of the site.

CSS/WYSIWYG

A CSS file containing the subset of CSS classes that will apply to content entered via a CMS WYSIWYG area must be provided. Please note that CSS with complex selectors will not render in the edit view of the WYSIWYG because the entire document context is not being provided. CSS for the WYSIWYG should be simplified to not rely on the full document context to render properly. However, the version of this CSS in the main CSS file(s) can include the complete set of selectors.

JavaScript and CSS Includes

JavaScript and CSS code should be provided in separate files, and brought in via the appropriate JavaScript "src" or LINK calls.

Comments
All code should be commented to improve comprehension during the CMS integration process, as well as assist with the eventual edits that will be made.

Wireframes/Site Map
Wireframes or Site Maps must be provided for any project to help your CMS team conceptualize the structure and function of the site.

Code Updates
Updates to delivered code must be appropriately commented and described to facilitate integration with the codebase. If possible, we will provide you with access to the CMS templates to make these updates directly**. Any updates to the code after it is delivered are considered a change to the scope and are billable.**

Layouts for All Pages
We must receive code/layouts for all pages that will have CMS managed code, including forms, search result pages, etc.

Link/HREF/SRC Values
The code must be structured to allow for absolute ("/") - not relative ("../") - link/HREF/SRC references for all elements. Exceptions to this rule are only available in rare extenuating circumstances, and will always be considered as a change from the standard scope. **Work required to accommodate relative link references will be billable.**

Recommendations

The following items are recommended, but not required, for your CMS project.

SEO/Accessibility
Images and other HTML elements should have ALT attributes with proper values where appropriate. The logic for what these values are and how they should be maintained must be communicated to the CrownPeak team for correct implementation in the CMS. Providing these values will assist with both SEO and Accessibility to disabled users.

CSS and Common HTML
We recommend that the CSS not aggressively modify standard HTML entity layout and display to facilitate the input of content. Bear in mind that the client will be updating the site for years to come and in most cases they are not HTML developers! Make your CSS open and flexible to withstand the rigors of time.

CSS – Inline Styles
Please avoid inline styles wherever possible: this adds complexity to the site, and might require more CMS templates than are needed to efficiently manage the content.

Valid Test Links
Ensure that linked content has valid URL's placed in each A HREF to ensure that it is rendering properly with the CSS provided.

Server-Side Includes/MasterPages

To facilitate updating of the content, we will abstract out various elements of the code that is not page specific to be drawn in via server-side includes (or similar mechanisms depending on the end hosting environment). If you are identifying includes during your development process, we ask that you coordinate with us on this so we can ensure that both teams are in sync on what code will be broken into includes.

Different Layout Versions of Pages

Pages that have different layout versions should have those different layouts specified in their code, and identified via comments. We should also receive separate pages so that we can see all those different layout versions. However, we will try to consolidate those into a single set of code that can be modified by conditionals to achieve the different layouts.

Coordination on Page Versions

We should coordinate on pages with similar function, but different look and feel within the site. For example, a news page might look different between the corporate area of the site, and the product area of the site, but have the same function. Where possible, it would be beneficial to unify the code of these pages so that we don't have to create multiple templates to perform the same function just because of code differences. Ideally, pages like this example would just use different CSS classes to adjust their look, and a simple conditional could provide the needed changes.

Other Components

The following items would be nice to have as components of your project, but are not required.

CSS and Image Usage

Where possible, try to avoid usage of images in the CSS that would require the client to directly modify the CSS. We can manage some of the CSS through the CMS (addition, ordering, removal of items), but this would be for items with pre-defined logic, and not freeform creation/modification of CSS code.

Using Scripting in Creation of Code

If possible, using scripting (ASP, PHP, JavaScript, etc.) to generate repeating elements in your code during your development process can be useful in identifying areas in which the HTML or CSS will need to be modified to support expansion/contraction of content. By identifying these issues during the initial code development process, we can leverage your expertise when creating CMS template code to streamline our efforts.

Related Elements in Pages

When designing/coding pages, please try to keep the content for each element together, instead of piecemeal. For example, think of a list of products with specifications to be printed out. You can assume that we will loop through the product data to print it out in the page. Our preference is that the code would be built to allow us to print a product and all its associated data in a single iteration of the loop, and then move on to the next product in the list - this is the most efficient way. We would rather avoid a setup where due to the way the code is built, we would have to print all the names of the products, then specification 1 for each product, then specification 2, etc. - this is less efficient as we have to process the same data multiple times.

Navigation Wrapper

Ideally, pages should be designed so that each has an identifiable navigation wrapper or "nav wrap". This is a concept that we employ in the development of CMS templates where the main navigation and page headers/footers are abstracted out into a single portion of template code that can be managed and updated centrally.

JavaScript for Site Functionality

Please note that while the CMS can pass through any type of server-side code (PHP, ASP, JSP, .Net, etc.), it cannot render this code in the CMS Preview mode. If there is dynamic functionality that you would like to have the ability to see in CMS Preview, creating that functionality