“WebFlavor” Course Template

Documentation – v. 2.0

Created: 06/09/2017

Last modified: 12/19/2017

By: Hauke Bahr

Contents

1. Introduction

The purpose and goal of this template is to provide a way to quickly create custom HTML courses that are versatile and flexible. Developers with a good understanding of the prerequisite skills (see below) will be able to create custom HTML courses that can accommodate any design, interactions, media objects, size and whatever other features the requirements specify. All those features can be added, modified, replaced or removed easily.

2. Prerequisite Skills

In order to modify the content of the course, the user needs to have an intermediate understanding of XML, HTML, and the Bootstrap CSS framework. Specifically, the user needs to know how to modify an existing XML document - including addition, removal, and changes of nodes – and how to modify HTML documents that are built upon the "Bootstrap" grid framework. Details will be described in this document.

3. Folder Structure

The folder structure is designed to reflect the separation of presentation and functionality and to support the simplification of making stylistic UI changes to the course.

```
course
|-- index.html
|-- imsmanifest.xml
|-- sco01.xml
|-- content
        |-- 0_0.html
        |-- 1_0.html
        |-- 1_1.html
        |-- ...

|-- resources
        |-- components
                |-- knowledgeCheck
                |-- accordion
                |-- ...
        |-- css
                |-- content.css
                |-- default.css
                |-- ...
        |-- js
                |-- api
        |-- media
                |-- images
                |-- audio
                |-- video
        |-- xml
                |-- settings.xml
                |-- glossary.xml
                |-- ...

|-- themes
        |-- theme X
                |-- css
                |-- fonts
                |-- img
                |-- js
        |-- theme Y
                |-- css
                |-- fonts
                |-- img
                |-- js
        |-- ...

|-- vendors
        |-- vendor a (e.g. bootstrap)
        |-- vendor b (e.g. jquery)
        |-- ...
```

4. Quick Reference

This section will go into a little more detail about the folder structure and explain which files to modify to make changes to an existing course or create a new course from the template.

The top-level files are:

`index.html`: This constitutes the UI shell of the course, it holds the navigation menu markup, and it initiates the course on the LMS and sets up several other things behind the scenes. Some changes to the html will need to be made to display the menu and navigation of the specific course you are creating. In future versions of the template, more of the menu and navigation will simply be set in the pertinent xml files but for now, html changes are required.

`imsmanifest.xml`: In this file, the course title needs to be set in five different places. In the template, you can simply search for "Official Course Title Goes Here" (main title as it will appear to the learners on the LMS launch page) and "official_course_title_goes_here" (unique identifier necessary for tracking progress, setting bookmarking and other things happening behind the scenes on the LMS) and then enter the official title of your own course in those five places.

> Note: The main title cannot contain any special characters or else the LMS might not accept it or display it incorrectly. The unique identifier cannot have spaces or special characters. Instead, use underscores to separate words. Conventionally, unique identifiers are spelled in all lowercase characters.

`sco01.xml`: Just like in **`imsmanifest.xml`**, search the document for "Official Course Title Goes Here" and "official_course_title_goes_here" and swap them for the actual course title, maintaining the specifications outlined above.

The **`content`** directory holds the pages that are loaded into the shell. This is where the markup and layout of each course page needs to be adjusted and any knowledge checks, interactives, and media need to be added.

`resources` holds all the files that comprise the core functionality as well as all the styles and media elements that a specific course requires.

`resources` → **`components`** holds JavaScript files that define functionality for course-specific components such as a glossary, custom video playback, knowledge checks, CTRs, accordions, etc. These are loaded into the pages in the **`content`** directory using the appropriate markup and code. Ideally, these should be re-usable but in practice, functionality requirements are often so different from course to course that custom code is usually unavoidable.

`resources` → **`css`** holds the course-specific style sheet **`course_content.css`** that defines style rules that are not shared across courses, i.e. styles at the page level. Examples would include page-specific placement and size of images, font variations, layout variations and anything else at the page level that is not covered by the general theme styles defined by the css files in the **`theme`** directory.

`resources` → **`js`** holds JavaScript files that comprise the core functionality of any course built with this template, such as initiating the LMS, bookmarking, injecting dynamic html from xml files, etc.

`resources` → **`media`** holds all media that a specific course requires. Each type of media has its own self-describing subfolder: **`audio`**, **`doc`**, **`img`**, **`and video`**. This directory is meant to hold course-specific media files only, i.e. audio files, documents, images, and videos loaded at the page level of a specific course. Files that are shared across courses (such as brand logos, universal resources, intro page videos, etc.) should be saved in the **`theme`** inside the **`themes`** directory.

`resources` → **`xml`** holds xml files that define specific elements of the course, such as the menu and navigation, the glossary, etc., as well as high-level settings that determine parts of the layout, content, tracking, etc. The goal of this template is to define most parts of the course using the xml files so that only a few necessary things have to be modified in the html and js files. However, as of this version, depending on the course requirements, the workflow is still a combination of some extensive html and js work combined with updating the xml files. The nodes in these files are mostly labeled to be self-explanatory. For a more detail explanation, see the section on xml below.

`themes` holds the folders that contain all the files that define the overall design of a course or group of related courses. For each new course or group of related courses that share the same design and elements, a new folder should be created inside **`themes`** and named according to the name of the course or group of courses, e.g. **`gm_fast`**, **`cigna_compliance`**, etc. Each of the specific course theme directories should contain sub-directories that hold the files necessary to change the course according to the requirements. At a minimum, this would be **`css`**, **`fonts`**, **`media`** (containing **`audio`**, **`doc`**, **`img`**, and **`video`** directories and mirroring the structure inside **`resources`**) and possibly **`js`** directories.

5. settings.xml

`settings.xml` is the heart of the template. It determines and controls a range of settings and features in the template and should be the first file to be modified (after the top level files `sco01.xml` and `imsmanifest.xml` as described above). This section will explain the nodes in `settings.xml` in detail.

`courseStorageID`: The settings and a range of other data of the template is stored in the local storage of the user's computer (similar to but not the same as cookies). `courseStorageID` is the unique identifier for the part of the local storage that holds the course data. Give this a unique name without spaces or special characters. The identifier is not case-sensitive. A good practice would be the client and the course name – e.g. "GM_Work_Vehicle_Technical_Specifications" – but it doesn't have to be.

`courseTitle`: The official title of the course as it will be displayed in the header (unless the header html is modified.

`courseSubTitle`: If the course has a subtitle, enter it here. It will be displayed underneath the header unless the header html is modified.

`version`: This refers to the version of the settings, not the course. Any time you make a change to the settings, you need to change the version number. It does not matter if it's ascending or descending, just as long as the number changes. The course will read the version number on startup and if it has changed, the local storage on the user's computer will be updated with the changes that were made to the settings. If the version number is not changed after a change was made to the settings, then the change will not be updated in the local storage and it will not be reflected in the course.

`theme`: This needs to be identical to the theme folder name that holds all the theme-specific css and other files. The course will read this node and automatically find the theme-specific files in the theme directory.

`cookieName`: If the course is not loaded from an LMS then the course progress (score tracking, last viewed page, etc.) is stored in a browser cookie. Enter a unique name that the course will use to create, read and update the cookie.

`completionMethod`: Set this to the desired completion method. The possible and valid values are stated and explained in `settings.xml` itself.

`menuPlacement`: Currently, the template allows the menu to be placed at the top or on the left. Alternatively, it can have no menu at all. Possible, valid values are stated and explained in `settings.xml` itself.

`hasSplashPage`: If the course has an intro or splash page that is not supposed to include the header or other elements that are visible on all other pages, this can be set to "true". The course will then display a full screen splash page at the beginning of the course. The markup for the splash page is defined in `modal_functions.js` in the `components` directory.

`hasGlossary`: If the course has a glossary, this node should be set to true. The markup for the glossary is defined in `modal_functions.js` in the `components` directory. Once the markup for the glossary is created, it can be opened by placing a button with the id `btnGlossary` in the menu.

`hasResources`: Currently not implemented but will open a page or popup with links to external resources in future iterations of the template.

`hasCards`: Several courses that were built with this template had the specific requirement to display "cards" or "content boxes" that included curated external links. The capability to display cards/content boxes was left in the template and can be activated by entering "true". Once the course reads that this setting is activated, it will look for and load **`card_content.xml`**. For details on how to set up pages with cards/content boxes in a course, see the section **`card_content.xml`** below.

`hasStrings`: Some courses have repetitive text in places such as navigation or instructions. Repetitive text strings can be defined in **`strings.xml`** and then loaded by setting this node to "true". For details on how to display strings in the course, see the section **`strings.xml`** below.

`bookmarking`: This was used in a course that had a launch page that branched off into different courses. The launch page kept track of which pages in which of the branched courses had been visited. Can be disregarded unless a similar setup is required.

`linkTracking`:

`pageCount`:

6. HTML Grid Layout – Powered by Bootstrap

Basics

The layout of the entire course is built as a grid of content elements that automatically arrange themselves according to the screen size of the device the course is being viewed on. The CSS grid framework Bootstrap is used to accomplish this.

> Note: The pertinent parts of Bootstrap that are used in the course will be described here. For complete documentation of Bootstrap, see http://getbootstrap.com/. For complete and specific documentation of the grid system utilized in the course, see http://getbootstrap.com/css/#grid.

Bootstrap arranges content elements in rows and columns, where each row can have a varying number of content elements divisible by 12. The rows and columns are defined and created by using specific HTML tags and CSS classes in the HTML document.

Bootstrap divides every row up into twelve columns. Each content element can take up any number of these columns. For example, you can have one content item that spans all twelve columns – i.e. the entire width of the content area – or you can have four content items, each taking up three columns, or three content items where two take up five columns and one takes up two columns or any other combination that is divisible by 12.

To illustrate this, let's look at some code examples:

Firstly, rows are represented in the HTML by `div` elements with the class name `row`. Rows need to be surrounded by HTML `div` elements with the class name `container`. This results in markup of this basic structure:

```
<div class="container">

      <div class=”row”>

            …

      </div>

      <div class=”row”>

            …

      </div>

</div>
```

In the above example, we have two empty rows that are set up to contain Bootstrap column content elements that can occupy between 1 and 12 horizontal "spaces" as described above. These column content elements are represented by HTML `div` elements with specific class names. The class names will determine how the column content elements flow on the page based on the available width of the viewport (i.e. the area of the screen in which the site/course is visible).

To illustrate how these elements work, let's first look at some of the class names:

```
”col-md-1”
”col-md-8”
”col-xs-6”
”col-lg-4”
```

The class names consist of three parts. The first part – **`”col”`** – simply makes this easily identifiable as a column element and all column elements start with it.

The second part can be one of four infixes: **`”xs”, ”sm”, ”md”, ”lg”`**. These define the screen width in which a certain element should occupy a certain number of column spaces (see below).

The third part is any number from **`1`** to **`12`**. This number defines how many column spaces a certain element should occupy.

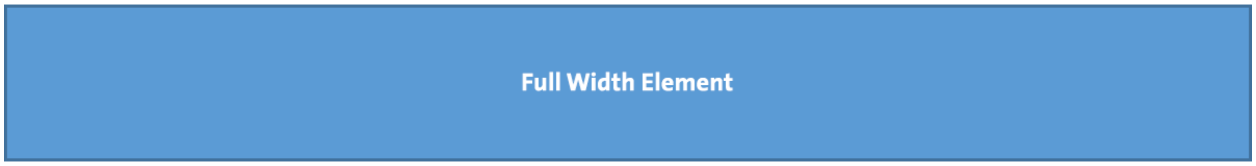
Let's look at examples:

```
<div class="container">

      <div class=”row”>

            <div class=”col-xs-12”>

            …

            </div>

      </div>

</div>
```

In this example, the markup defines one column that will occupy all 12 column spaces – i.e. the entire available width of the container – in all screen widths. It will span the entire available container width on large computer screens down to narrow devices like cell phones.

> Note: The look of column content elements is not determined by these class names. Separate CSS is required for that. The class names only determine the layout.

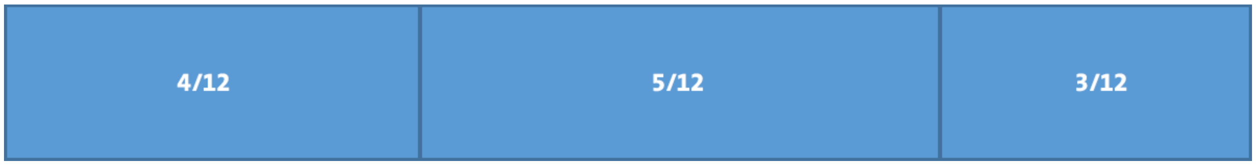
Disregarding additional CSS that would define the design of the elements, the above markup can be represented as the illustration below:



```
<div class="container">

      <div class="row">

            <div class="col-lg-4">

            …

            </div>

            <div class="col-lg-5">

            …

            </div>


            <div class="col-lg-3">

            …

            </div>

      </div>

</div>
```

In this example, the markup defines one row with three column elements. The first element will occupy 4 column spaces, the second 5 column spaces, and the third 3 column spaces. Note that they add up to 12. The **`"lg"`** part of the class name defines that the elements will occupy 4, 5, and 3 column spaces respectively on **only** large screens. As soon as the screen width falls below a certain value, each of the 3 elements will occupy the full width of the container.

Disregarding additional CSS that would define the design of the elements, the above markup can be represented as the illustration below:



Note: Bootstrap will add gutters (horizontal and vertical spaces) between the elements. They are not displayed here for simplicity.

Keeping in mind the examples above, let's look at the specific viewport width values that are defined by the **`”xs”, ”sm”, ”md”, ”lg”`** parts of the CSS class names.

`xs:` No specified width - the element will be displayed occupying the number of column spaces specified by the number part of the CSS class on all screens, from large computer screens to "extra small" device screens.

`sm:` 768px – the element will be displayed occupying the number of column spaces specified by the number part of the CSS class name as long as the viewport width is 768px and above. Typically, this includes tablets and larger devices in the landscape position but not smaller tablets and cell phones. On screens of 767px and below, the element will occupy the entire width of the container. At this screen width range, Bootstrap will set the width of the container element to 750px.

`md:` 992px – the element will be displayed occupying the number of column spaces specified by the number part of the CSS class name as long as the viewport width is 992px and above. Typically, that only includes computer screens and some oversized tablets. On screens of 991px and below, the element will occupy the entire width of the container. At this screen width range, Bootstrap will set the width of the container element to 970px.

`lg:` 1200px – the element will be displayed occupying the number of column spaces specified by the number part of the CSS class name as long as the viewport width is 1200px and above. That only includes large computer screens. On screens of 1199px and below, the element will occupy the entire width of the container. At this screen width range, Bootstrap will set the width of the container element to 1170px.

Combining Class Names

The class names described above can be combined to get even more control over how content elements flow on the page across different devices.

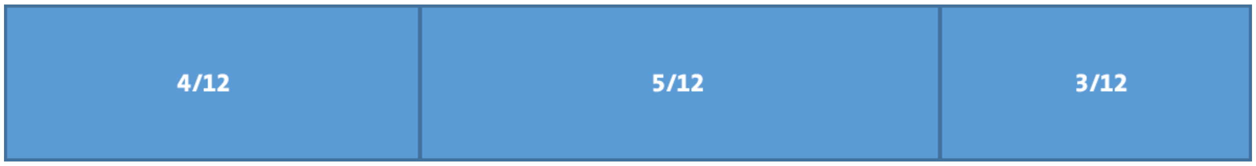
Example:

```
<div class="container">
     <div class=”row”>
          <div class=”col-lg-4 col-md-4 col-sm-3 col-xs-12”>
          …
          </div>
```

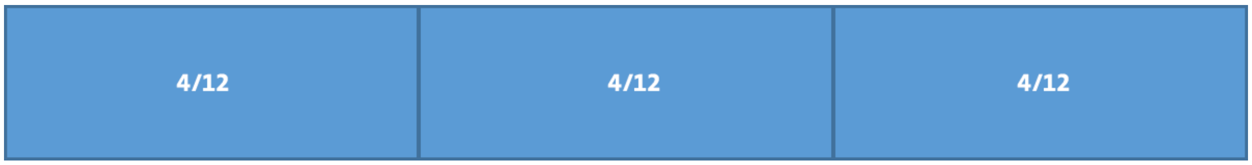
```
            <div class=”col-lg-5 col-md-4 col-sm-3 col-xs-12”>
            …
            </div>

            <div class=”col-lg-3 col-md-4 col-sm-3 col-xs-12”>
            …
            </div>
      </div>
</div>
```

Disregarding additional CSS that would define the design of the elements, the above markup can be represented as the illustrations below, depending on the viewport width:

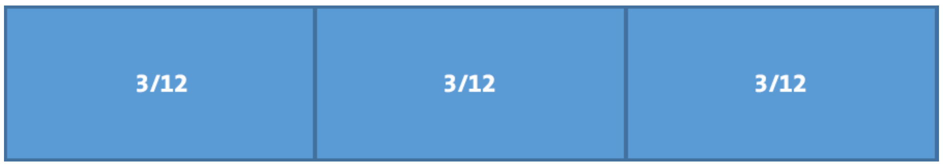
Large screens (1200px wide and above):



Medium screens (between 992px and 1299px width):

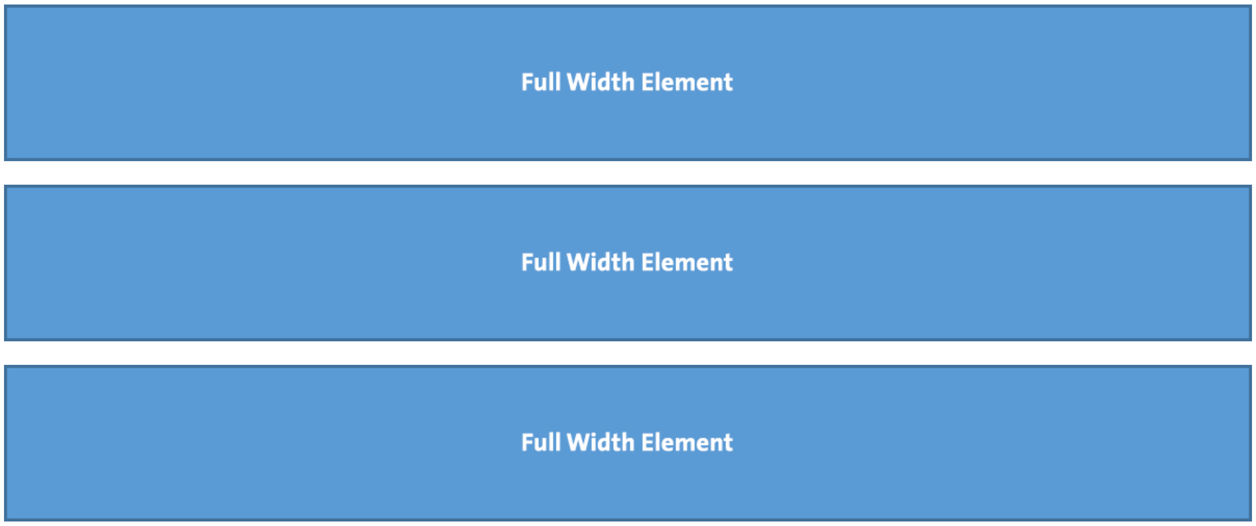


Small screens (between 768px and 991px width):



Note: If the specified combined width of content elements does not add up to 12, the elements will not fill the entire width of the content area. In this example, each element occupies 3 of the total available 12 spaces or columns so they only occupy a combined 9 spaces, leaving a gap at the end.

Extra small screens (between 0px and 767px width):



Note: Since each element is defined to occupy the entire width of the content area on extra small screens, they will flow into a stacked layout.

Many other layout configurations can be accomplished using the Bootstrap grid (e.g. nested grids, offsets) but the markup explained in this section covers the basics. For more information, please see the full Bootstrap documentation linked above.