Creating an example responsive Web3D Mobile App

# Introduction

This tutorial requires no previous knowledge of Dreamweaver, HTML, CSS, JavaScript, JQuery, AJAX, JSON or MVC (these technologies will be briefly explained in the technical background section) — we assume you are a beginner! As such, the tutorial will take the approach of **assembling** an example Web3D Mobile App from components created specifically for this workshop.

These components created for this workshop are based in part on code (CSS and JavaScript) and contents developed for a recent AHRC Beyond Text project called Reanimating Cultural Heritage (RCH) where we built a cultural object repository and web site that aggregates about 3,000 objects from several museums including the British Museums and British Library (see [www.sierraleoneheritage.org](http://www.sierraleoneheritage.org)).

The components created for this workshop are not perfect and are meant simply to illustrate the concept of responsive web design — it’s a dead cert that there will be many more elegant ways of achieving the same result, so we apologies for any rough edges!

However, first we will cover some background to set the scene, we’ll try to be brief!

## Teaching and learning scenario

We proposed this workshop at very short notice, about 3 or 4 weeks ago without having a concrete brief about what the workshop should cover. The original email from the Sussex library had a subject title of **‘Using AR in teaching and learning’** and asked for a brief 15-minute talk **‘… about some of the innovative things that are happening within Informatics, particularly around the use of AR’**.

Assuming this tutorial is taken later in the workshop a couple of speakers from our research group (Interactive Systems) will have already given you a brief introduction on their research work around Technology Enhanced Learning, but we thought it would be interesting to try and run a short (about 3 hours) hands on workshop (including demonstrations) to try and build a simple Web3D Mobile App that illustrates the use of 3D and AR in some learning context. This tutorial is one component of that workshop, rather hurriedly put together in the last two weeks — T this point in time (Sunday before the workshop starts) it is clear that we won’t be able to complete all parts of this particular tutorial, but if we can get through Part 1 and have a look at the results for Part 2 and 3, we think that will illustrate the concepts quite well — you will be able to finish the remaining parts in your own time if you wish.

To create the Web3D Mobile App in this tutorial we need some kind of teaching and learning scenario, which also has access to some existing data (contents) that could be adapted for the app. It is difficult to ‘dream up’ the data from scratch in such a short time; ideally we need some data that can be augmented with additional media contents (e.g. some 3D objects, video, etc. for use in an AR application context). To this end, as indicated above, we have access to some 3,000 cultural objects from our RCH repository. So it seems sensible to create (see the content creation tutorials and demos you completed previously in the workshop) additional media contents and integrate this new content with the existing cultural object data for presentation in a **Web3D Mobile App** and an **Aurasma AR Magic Book**.

We hope you will be able to see beyond this cultural object based Web Mobile App and Aurasma ARMagic Book example to create your own examples with your own media objects, etc. The simple example Web3D Mobile App and Magic Book created in this tutorial will present online (on browser, tablet and smartphone) a small collection of museums’ digital artifacts (for simplicity in this tutorial we will limit the number of cultural objects handled by the app to 4 or 5) represented by a set of associated media objects (3D, AR, Image, Audio, Video, Text and Metadata). However, with some extra coding it could be expanded.

Given the short time we have available it will not be possible to create every element of the Web3D Mobile App from scratch, so for expediency we will use the approach of ‘here’s one we made earlier’ at appropriate development stages. We envisage this particular tutorial being completed in 3 stages:

1. **Part 1 — Building fluid grid layouts in Adobe Dreamweaver CS6** into which you will insert some static media contents to test the resulting responsive Web3D Mobile App. Part 3 will insert dynamic contents
   * Once you understand the concept or if you run short of time you can switch to ‘here’s one we made earlier’. The tutorial files have a folder called **Completed Part 1**

If you only complete this part of the tutorial in the 45 minutes allotted, that will still have illustrated the concept of how to make your app responsive to different devices and screen layouts. You can then take a look at the final results for Part 2 and 3.

1. **Part 2 — CSS Style the fluid grid layout** in Part 1 based on existing code elements. You will insert new styling <div> blocks from existing code (into the fluid grid <div> blocks
   * Again, once you understand the concept, you can switch to ‘here’s one we made earlier’
2. **Part 3 —** **Add dynamic update to the styled responsive fluid grid layout** in Part 2. You will replace the hardwired data with dynamic data, i.e. insert the JQuery code and access the JSON file (data store)
   * Again, once you understand the concept, you can switch to ‘here’s one we made earlier’

It is highly unlikely you will get past this point in the workshop, so we suggest you might like to skim read from here and quickly move to **Part 1 — Building fluid grid layouts in Adobe Dreamweaver CS6**.

1. **Part 4 — Add more objects to the Web3D Mobile App data store**. Here you encouraged to examine the JSON object file and add a new object, perhaps the 3D object that was created during the 123D Catch demonstration. You will need to modify the code slightly to use the new object.
2. **Part 5 — Connect the Web3D Mobile App to the Sierra Leone Heritage repository**. Unfortunately, this is well beyond the scope of this workshop, but to give you an idea of what would be required, you would need to:
   * Modify the JQuery code to initiate request to a Sierra Leone Heritage web service API and process results return. The web service API would perform API requests on data in the Sierra Leone Heritage database that utilize SQL queries on the database. For example, your original request may have been a keyword ‘search’ for objects of a certain type, or ‘browse’ a particular collection.
     + This assume we already have a web service API available that could be used to query the database. However, while we do have PHP code that queries the database, it is not specifically configured as a web service at the moment, it returns HTML rather than a JSON object. Nevertheless, this PHP code could be used as the starting point to create a Sierra Leone Heritage web service API)
   * Your ‘search’ or ‘browse’ request for cultural objects would then return a JSON file (object) that you would read as you have already done in Part 3 and 4. However, now you have an array of objects, but you don’t know how many objects are returned, so you would probably have to count the length of the array and use this statistic to build a dynamic gallery that showed thumbnails of each object. This would replace the ‘fixed’ Browse Objects function.
     + Of course, if you were building a specific application based on a known number of cultural objects, you could simply pre-load a gallery, at the moment your Web Mobile Application is only handling 4 or 5 objects. So clearly the Browse Object function has to be modified to handle a random number of objects.
     + Also, there are countless open source JavaScript galleries that could be adapted for this task.
   * There are probably several other things to consider, but we haven’t had the time to implement this yet, so we leave this to your imagination.

Finally, we have constructed this tutorial so that you can hopefully complete Part 1, in this workshop, maybe even get into Part 2 and 3, but will likely need to finish off line in your own time. To aid in this we have supplied completed versions of Part 1, 2 and 3.

## Technical Background

**Skip this section, read it later if you are short of time!**

It might be instructive to briefly describe the technologies we will utilize in developing this simple Web3D Mobile App, feel free to skip this section and go straight to ‘**Building fluid grid layouts in Adobe Dreamweaver CS6**’ if you just want to dive in given that we have limited time in the workshop.

### Approach

In this tutorial we will utilize several technologies, but we also point out that there are many other ways to develop mobile apps. In general we could consider three approaches:

1. Use a responsive web design approach that utilizes a fluid grid layout for HTML5. There are many frameworks available that aid in responsive web design, but they usually rely on a fluid grid layout where element or block widths such as columns are expressed as a percentage of the page width. For example, Dreamweaver has a fluid grid layout scheme based on JavaScript and CSS3, other frameworks include Response based on JQuery, Bootstrap, Foundation, and AngulaJS to name a few. The key thing is that responsive web design allows you to create your web app and deliver it on different devices, e.g. browser, tablet and smartphone, where elements (sections, images, etc.) of the page scale and rearrange (due to the fluid grid layout) to the device screen size.
2. If you know that you only want to target a specific mobile device, e.g. an Android smartphone or iPhone, you could use a particular development methodology to optimally target that device. For example, you might use the ADT (Android Developer Tools) plugin for the Eclipse IDE (Integrated Development Environment). These development tools are soon to be replaced with Android Studio. If your app needs to run fast, or access the device’s hardware, and so on, this may be the best approach.
3. There are also abstraction frameworks, for want of a better term, that allow you to design at a higher (HTML/CSS) level. For example, PhoneGap is an open sourvce framework that integrated with Dreamweaver, which allows you to create mobile apps using standard web technologies: HTML, CSS, and JavaScript. Dreamweaver also offers app developer layouts and templates to aid in development of specific device apps. For example, they have a JQuery Mobile template to get started. There are many other frameworks such as Sencha that can you to deploy HTML5 applications on desktops, tablets and smartphones.

Approach 2 and 3 are beyond the scope of this workshop, so we will use approach 1, and in particular utilize Dreamweaver and its fluid grid layout. We will also utilize JQuery libraries without the Dreamweaver specific JQuery code hints. It is interesting to point out that a recent article

### Technologies

Before we get started on this ‘Building fluid grid layouts in Adobe Dreamweaver CS6’ tutorial let’s take a quick look at the technologies we will be using.

Unfortunately, we haven’t had the time to write our own succinct description or definition of these technologies, so we provide links (several are links to ‘[tutorialspoint, simply easy learning](http://www.tutorialspoint.com/index.htm)’, but we don’t specifically vouch for them, please explore yourself) for you to allow further reading, you can also, of course, Google these search terms to find many interesting articles online. One interesting article points out that the “majority of ‘native’ mobile apps written in 2014 contain significant amounts of HTML and JavaScript” [[readwrite](http://readwrite.com/2014/11/17/html5-javascript-everywhere-mobile-tom-dale-emberjs)]

Some links connected to the TutorialsPoint Simply Easy Learning site.

* + [HTML5](http://www.tutorialspoint.com/html5/index.htm)
  + [CSS3](http://www.tutorialspoint.com/css/index.htm)
  + [JavaScript](http://www.tutorialspoint.com/javascript/index.htm)
  + [JSON](http://www.tutorialspoint.com/json/index.htm)
  + [MVC](http://www.tutorialspoint.com/struts_2/basic_mvc_architecture.htm)

Some links connected to the JQuery Learning Centre.

* + [JQuery](http://learn.jquery.com/)
  + [AJAX](http://learn.jquery.com/ajax/)

Ever wondered what an API is?

* + [API](http://readwrite.com/2013/09/19/api-defined)

Some links to world of 3D.

* + [3D](http://www.youtube.com/watch?v=MeVaCOZ62So)
  + [X3D](http://www.web3d.org/getting-started-x3d)
  + [X3DOM](http://www.x3dom.org/)

Finally, here’s a nice link to LayerAR, which shows a video (Design Fiction) of AR being used in everyday life.

* + [AR](https://www.layar.com/)

# Part 1 — Building fluid grid layouts in Adobe Dreamweaver CS6

This tutorial is adapted from the ‘Building fluid grid layouts in Adobe Dreamweaver CS6’ by Tommi West. As such, we will effectively utilize Tommi West’s Adobe tutorial in its entirety, but acknowledge all IP belongs to Adobe. Here is the link to the original tutorial:

* <http://www.adobe.com/inspire/2012/08/fluid-grid-layouts-dreamweaver-cs6.html>

For convenience we have created this ‘print out’ of the tutorial so that you can better deploy your screen resources while working in pairs (one participant can read/instruct while the other participant can use the tools, i.e. Dreamweaver). Further, by necessity, we have modified the tutorial to fit this workshop’s Web3D Mobile App requirements.

**Note: verbatim or paraphrased text from the original Adobe tutorial is indented, while our modifications are coloured in GREEN.**

With the ever-increasing use of mobile devices, it has become necessary to design online content that appears on multiple screen sizes and a myriad of desktops, laptops, tablets, and smartphones. The challenge involves designing sites that adapt to fit a variety of different resolutions and use the available screen real estate effectively. Previously, these goals have proven to be time-consuming. Hand-coded media queries and complex mathematical calculations were often required to deliver online presentations with fluid layouts to reach the widest audiences.

Adobe Dreamweaver CS6 introduces the Fluid Grid Layout feature to help make designing for multiple screens easier. Fluid grid-based layouts display content in containers that have widths set to percentages, which react proportionately to changes in screen sizes. In this article, I show you how to build fluid grid layouts that address these challenges.

To follow along with these instructions the required sample files are already saved onto your desktop. In a folder called Web3D\_Mobile\_App.

## Defining a site

If we are using an HTML web-authoring tool, as we are, such as Dreamweaver, one of the first things we need to do is to define a site in a local folder on your development machine. In this case your development machine is a PC in Informatics Lab 4, but could just as easily be your own laptop if you have the tools available. Note, you can download a 30 trial of Dreamweaver.

Follow these steps to define your site:

1. Launch Dreamweaver CS6 and choose Site > New Site.  
     
   You will find Dreamweaver either as a shortcut on the desktop or in the Start folder, bottom left of the screen. Ignore the Welcome Screen for a minute. We have created a folder on the desktop called Web3D\_Mobile\_App that holds all the sample files you will need.
2. In the Site Setup dialog box, type Web3D\_Mobile\_App in the Site Name field (see Figure 1).
3. Click the folder icon next to the Local Site Folder field, ringed in red.
4. In the Choose Root Folder dialog box, navigate to the Web3D\_Mobile\_App folder on your desktop. Click Select (Windows) or Choose (Mac OS).

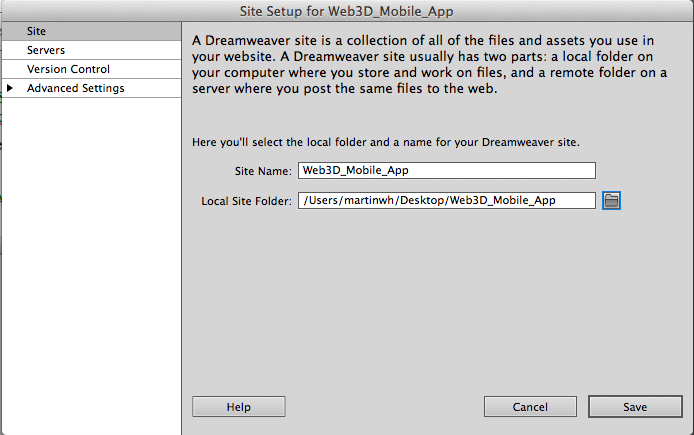


Figure 1. Create a site definition to specify the local root folder for the project.

1. Click Save to close the Site Setup dialog box.

## Creating a fluid grid layout

Follow these steps to create a fluid grid layout:

1. You can choose Fluid Grid Layout from the Welcome Screen, or if this has disappeared then: Choose File > New Fluid Grid Layout. Alternatively, you can choose File > New and then choose Fluid Grid Layout in the New Document dialog box (see Figure 2).

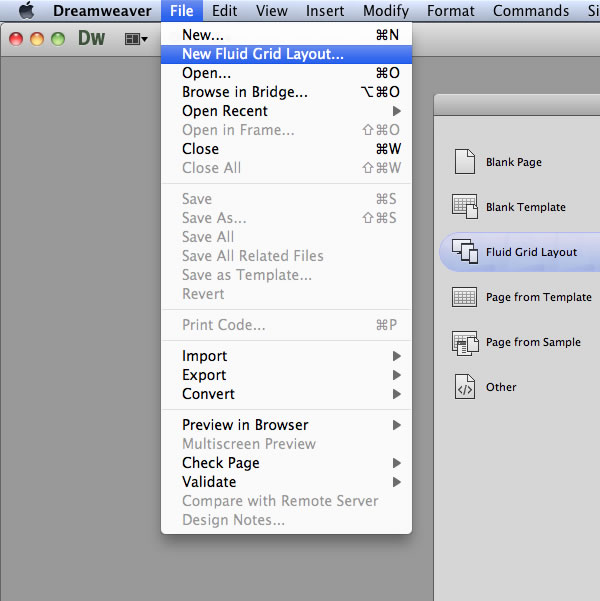


Figure 2. Create a new fluid grid layout.

You can use the dialog box that appears to define the layout settings for your project. The grid that you'll use to lay out containers is based on the number of columns you specify for each screen. By default, these values are set to 5, 8, and 10 columns for mobile devices, tablets, and desktop screens respectively.

1. Update the Desktop layout to use 12 columns instead of 10 by typing directly in the field (see Figure 3). For this Web3D Mobile App project, you can leave all the other default settings, except the gutter width, see below.

Notice the default targeted widths — Mobile: 91%, Tablet: 93%, and Desktop: 90%. These percentages are also editable.

The Percentage of Column Width specifies the width of the gap that appears between the columns, also known as the gutter value, as a percentage. By default, the layout gutter value is set to 25% of each column width. If you edit the gutter value field, the percentage you enter affects all three-screen resolutions.

However, if I remember correctly I reduced this to 15% in the example Web3D mobile App. It is really an aesthetic thing, but is fixed once you choose. In reality because you will have different numbers of columns for each screen layout the actual width will vary between screens (smartphone, tablet and browser) the actual gutter width will vary, but total screen real-estate will be whatever you set fort eh gutters between columns (if I understand it correctly) Unfortunately, you can’t change this value when you leave this screen. This gutter value is the width between each column and Dreamweaver uses it to make some complex calculations related to the column layout of your application. I think I set my Web3D Mobile App gutter width to 15%, you can leave it at 25% or change it. Figure 3 shows the gutter width set to 15%.

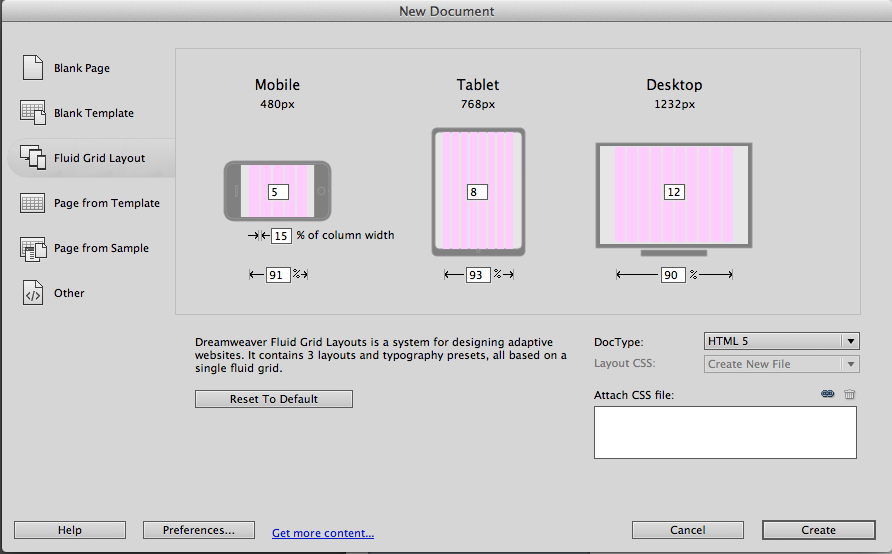


Figure 3. Set the number of columns for the Desktop layout to 12.

1. Click Create. In the Save Style Sheet File As dialog box that appears, navigate to the css subfolder located inside the Web3D\_Mobile\_App folder on your desktop.
2. Name the layout **mobile\_web.css** and click Save. The dialog box closes, and the layout is displayed in Design view. Before continuing, it is important to save the HTML file.
3. Choose File > Save. In the Save As dialog box, save the file in the Web3D\_Mobile\_App folder on your desktop. Name the HTML page **index.html** and click Save.

The Copy Dependent Files dialog box appears. Fluid grid layouts require two different scripts to function correctly. The dialog box lists these files:

* + **boilerplate.css:** An HTML5 boilerplate that includes styles you can use to get started
  + **respond.min.js:** A JavaScript file that adds support for fluid layouts to older browsers

You must save these two dependent files in the site folder that contains index.html. They can be in a subfolder. Be sure not to move or rename these files after you copy them. When you upload the site, you'll need to put these dependent files on the host server with the other site files.

We will put these files in the scripts folder. When you upload the site, you'll need to put these dependent files on the host server with the other site files.

1. Click Open (the icon next to the path in ‘Copy files to:’) to select the scripts subfolder. Then click Copy to copy the dependent files. The Copy Dependent Files dialog box closes, and the layout is displayed in Design view again.

At this point you will have copied the boilerplate.css file into the script folder, and while the original tutorial saves don’t move it, you can actually move the boilerplate.css file into the css folder leaving the respond.min.js file in the scripts folder. When you do this Dreamweaver will show the update dialog box, so click update. This will then update the path in the HTML code to access the boilerplate.css code in the css folder.

At this point, you have a folder (Web3D\_Mobile\_App) on your desktop that contains all the site files you need to build the fluid layout for three screen resolutions.

## Inserting fluid grid layout div containers

Follow these steps to insert the div containers that hold the content for each region of the layout:

1. Click Split at the top of the Document window to see Code view and Design view simultaneously. In the Title field, name the page **Sierra Leone Mobile**.
2. In Design view, notice that one fluid grid layout div named LayoutDiv1 is included in the layout by default. Click once inside the default div container to make it active, and then click the tag selector at the bottom of the Document window to select the entire div container, including the opening and closing tags.
3. Press Backspace (Windows) or Delete (Mac OS) to delete the default div (see Figure 4).

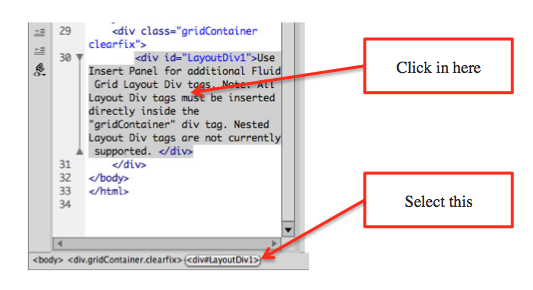


Figure 4. While the default div container is selected, delete it.

1. In Code view, verify that the insertion point appears after the opening <div> tag. This sets the placement of the div you'll insert in the next step, so it will nest inside gridContainer.
2. Open the Insert panel (Window > Insert). Choose Layout from the pull-down menu at the top and select Insert Fluid Grid Layout Div Tag. The Insert Fluid Grid Layout Div Tag dialog box appears.
3. Name the new container **header**. In the dialog box, notice that the Start New Row option is enabled by default (see Figure 5).

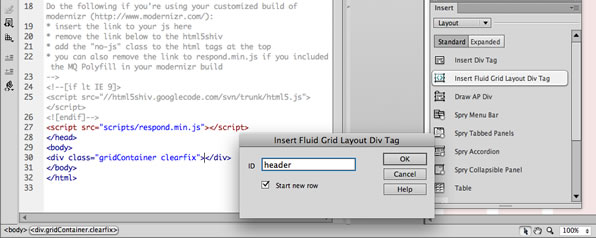


Figure 5. Name the new div container and leave the Start New Row option enabled.

When the Start New Row option is enabled, the content in that div container will not appear in a column; it will always appear on its own horizontal row of the screen, regardless of the visitor's screen resolution.

1. Click OK to insert the new Header container and close the dialog box.

Note: Always ensure the new div containers you insert are nested within the outer div container named gridContainer.

After you insert the first div, the insertion point is positioned after the closing </div> tag of the header div container.

1. Repeat steps 5–7 to insert another div container named **objectPageSelector**. Leave the Start New Row option enabled for this div container as well.

The layout now displays two div containers (header and objectPageSelector) nested in an outer container named gridContainer. The two layout containers hold placeholder text, but don't worry about that for now. Later you'll replace that text with site content for your Web3D Mobile App.

In Design view, look at the left side of the layout and notice that both the header and objectPageSelector containers are vertically aligned (see Figure 6). That is a visual indicator that they are both set to appear on a new row when viewed. Additionally, they are both set to span the entire width of the grid columns. By default, new div containers you insert are set to 100% width.

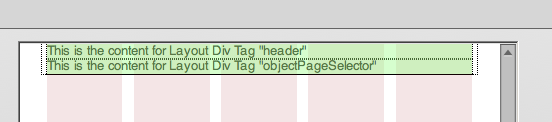


Figure 6. The header and objectPageSelector containers are aligned vertically and span the entire grid.

1. In Code view, verify that the insertion point is positioned after the Nav container's closing </div> tag to insert the next div container.
2. In the Layout section of the Insert panel, select Insert Fluid Grid Layout Div Tag. The Insert Fluid Grid Layout Div Tag dialog box appears.
3. Name the new container **mediaObjects**. This time, disable the Start New Row option by deselecting the checkbox. Click OK.

In Design view, notice that the mediaObjects container looks slightly indented on the left side. This is a visual indicator that the Start New Row option is disabled for the Teaser container.

1. Repeat steps 9–11 to insert three more div containers. Name the first new container **objectMediaSelector**. Name the second new container **objectDescription**, and the third new container **objectData**. As you insert each one, deselect the Start New Row option before clicking OK.
2. Insert one final div container. Name it **footer**. This time, use the default setting to keep the Start New Row option enabled.

When you look at the layout in Design view, you'll see a total of seven div containers. The top two (header and objectPageSelector) are set to appear on new rows. The next four (mediaObjects, objectMediaSelector, objectDescription and objectData) are set up to possibly appear as columns (space permitting). And the last div container, footer, is set to appear on a new row. The indenting on the left side indicates that the mediaObjects, objectMediaSelector, objectDescription and objectData containers could be displayed side-by-side (see Figure 7).

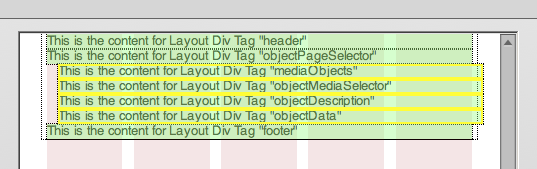


Figure 7. The Teaser, About, and Article div containers are indented on the left side.

1. Choose File > Save All. This ensures you save both the HTML file and the CSS file.

## Populating the layout with site content

In this section, you'll replace the placeholder content with some HTML to simulate the content of the final Web3D Mobile App site:

1. Click Split to see both the Code view and Design view panes of the Document window.
2. In Design view, click inside the header div. Select the placeholder content by pressing Control+A (Windows) or Command+A (Mac OS) and then delete the contents by pressing Backspace (Windows) or Delete (Mac OS).
3. In Code view, notice that the header div is now empty, and the insertion point is located between the opening and closing <div> tags. Click that location in Code view to set the location to paste the new content.
4. Choose File > Open. Navigate to the Web3D\_mobile\_App folder on your desktop and locate the file named content.txt in the content subfolder. Click Open.
5. Select the first section of content below the text that says **Content for header**. Copy the selected content by pressing Control+C (Windows) or Command+C (Mac OS).
6. Click the tab at the top of the Document window to return to index.html. Without clicking anywhere else on the page, press Control+V (Windows) or Command+V (Mac OS) to paste the content into the header div (see Figure 8).

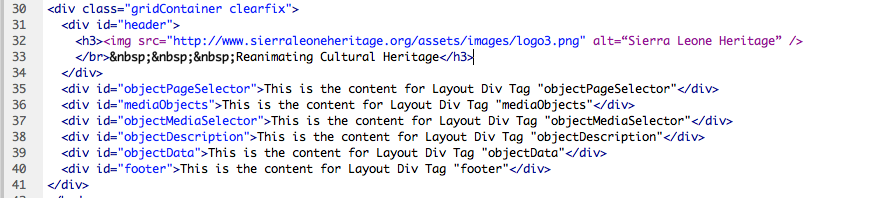


Figure 8. Paste the HTML code inside the header div container to populate the region.

**Note**: During this process, verify that the pasted content is inside the header div container, rather than outside its closing div tag. Be sure not to accidentally delete the opening and closing div tags while editing the code. Also, feel free to tidy up and indent the code.

1. Repeat steps 2–6 to replace the placeholder text with site content in the mediaObjects, objectMediaSelector, objectDescription and objectData and footer div containers.

When you build fluid grid layouts, you define three unique layouts to target each screen resolution. The mobile-web.css file adapts to the available screen size as the content is viewed on different screens (or when the browser window is resized).

## Building the layout for each screen size

To build the layout for each of the three target screens, you need to start with the mobile layout and then update the container positioning for the tablet and desktop layouts. Using this strategy, you can customize the way the div containers appear on different screen resolutions.

Dreamweaver CS6 includes the new Resolution Switcher that enables you to quickly jump between the mobile, tablet, and desktop resolutions. By default, the Resolution Switcher is set to Mobile Size. You can click the icons along the bottom of the document window to switch back and forth at any time as you build the layout (see Figure 9).



Figure 9. The Resolution Switcher is set to Mobile Size by default.

1. In Design view, select the third div container from the top, named mediaObjects.

As you view the selected container, notice the handles on the left and right side. In some programs, you can drag the handles on either side to resize an element. However, these handles control specific settings:

* + Drag the handles on the left side of div containers to set the margin.
  + Drag the handles on the right side of div containers to scale the width.

The Resolution Switcher is currently set to Mobile Size. If you want to make any changes to the mobile layout, you can edit the containers while this setting is active. However, for this project, keep the containers as they are positioned by default; in the smallest screen, each container appears in its own row, stacked vertically without any columns.

1. Use the Resolution Switcher to select Tablet Size (768 x 1024). Design view updates to display a wider layout with eight columns instead of five.
2. Select the third div container from the top named mediaObjects. When you select it, a blue border appears around it. Drag the right handle to the left, resizing the mediaObjects div container until it only spans the first four columns (see Figure 10).

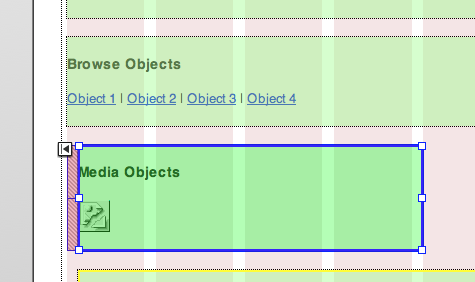
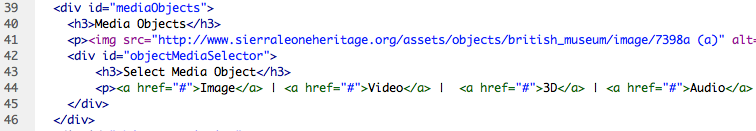
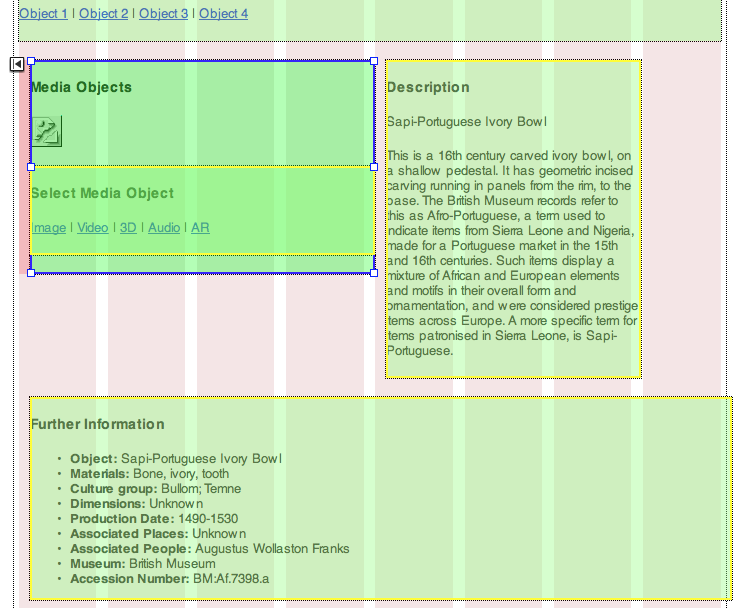
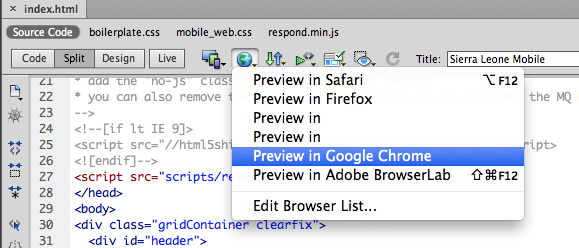


Figure 10. Drag the right handle to decrease the width of the mediaObjects div container.

The div container snaps to the closest column in the grid as you resize the width. In this example, the edge snaps to the left side of the fifth column. The grid system makes it easy to keep the containers lined up.

**Note:** Remember that the left handle increases the margin, rather than adjusting the width of the container. It's important to drag the correct handles as you build layouts.

1. Select the fourth container, named objectMediaSelector. Resize the div by dragging the handle on the right side to the left. As the objectMediaSelector container becomes more narrow (e.g. 3 columns wide, it jumps up next to the other mediaObjects container. We don’t really want this to happen.   
     
   At this point we would like to ensure that the objectMediaSelector container stays below the mediaObjects container, because in our final Web3D Mobile App the objectMediaSelector will be used to select media objects (image, 3D model, video, etc.) to be displayed in the mediaObjects container. It would look odd if it floated round to the next right hand column. To achieve this we need to move the objectMediaSelector container inside mediaObjects container. Probably the easiest way to do this is to delete the objectMediaSelector container and insert a new objectMediaSelector container. Place the cursor after the </div> of the mediaObjects container and insert a new Fluid Grid Layout Div Tag called, you guessed it, objectMediaSelector. As before, insert the contents for the objectMediaSelector.  
     
   Your HTML code should look like that in Figure 11, with a bit of tidying up.  
      
     
   Figure 11. Insert the mediaObjects and objectMediaSelector another <div> tag container  
     
   Switch back to the Design view and select the right handle of the objectDescription container and resize to three columns so that it jumps up to alongside the mediaObjects and objectMediaSelector containers on the right hand 5th, 6th and 7th columns. You should have a layout like that shown in Figure 12.  
     
     
     
   Figure 12: Resize the objectDescription container to three columns so that it jumps up  
     
   In the mediaObject container top left hand side you will see the a small alignment handle used to align the container with the grid, I’ve circled it in red. Click on this to align the container. Now you can resize the objectDescription container to cover the 4 right most grid columns.  
     
   Next resize the objectData container to 4 columns. Your tablet layout will now look like that shown in Figure 13. When shown in a web browser the objectDescription and objectData containers will float round to the right of the mediaObject container. If you click on the Preview/Debug in browser button, see figure 14, you can see the results so far are fluid in smartphone and tablet screen sizes.  
     
     
     
   Figure 14: Preview the app so far in a Google browser.

The results should look something like that show in Figure 15. Here we can see the smartphone layout is composed of a single column and the tablet is laid out as two columns, and we haven’t configured the desktop layout yet, that the next job.

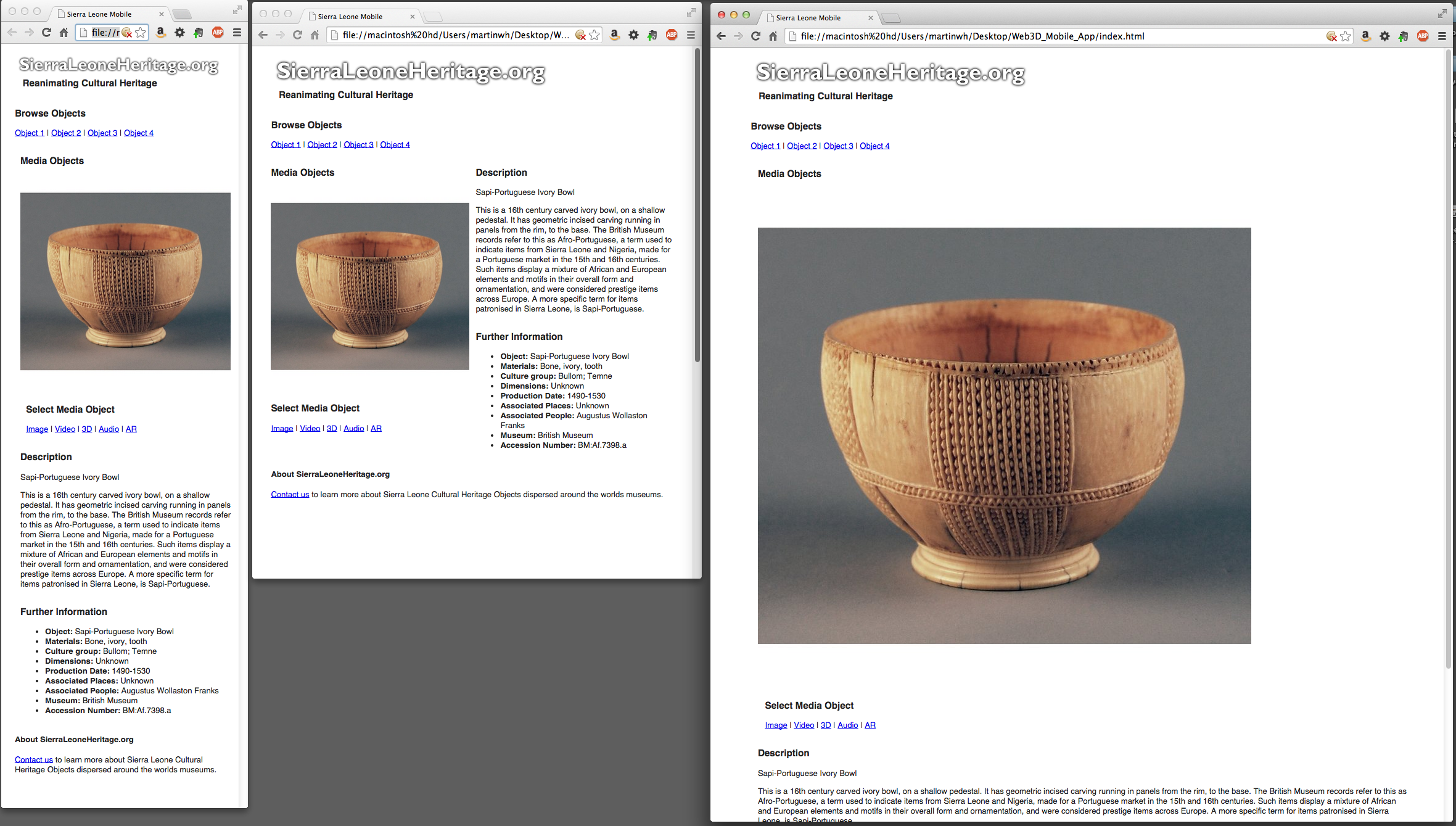


Figure 15. Our responsive results so far.

Use the Resolution Switcher to select Desktop Size. Design view updates to display a wider layout. The grid now contains twelve columns instead of eight.

1. Select the mediaObjects div. Drag the right handle to the left until the container's width becomes so narrow that it spans the first four grid columns. Notice the objectMediaSelctor resizes with it. Click on the mediaObjects alignment tab to align with the objectPageSelector container above.
2. Select the objectDescription div. Resize it using the right handle until it spans the three middle columns on the same row as the Teaser div.
3. Select the objectData dive. Drag the right handle to the left until the container's width becomes so narrow that it jumps up to the same row as the mediaObjects and objectMediaSelctor div containers.

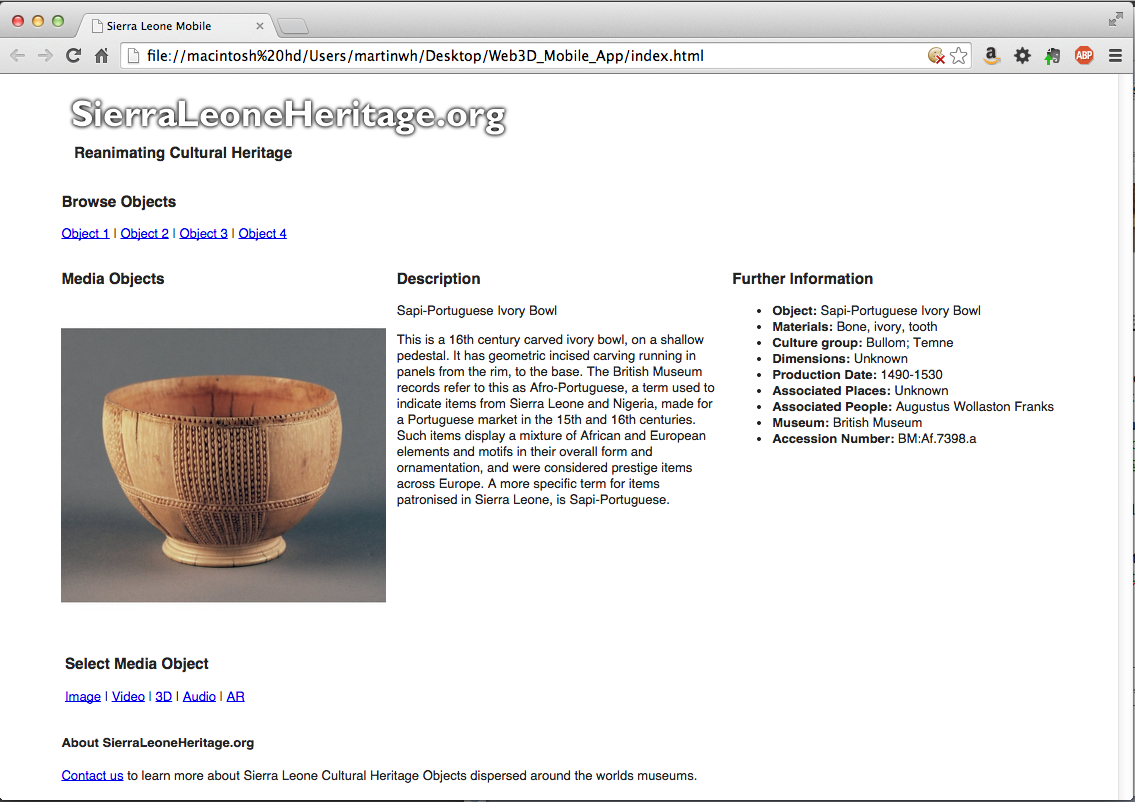
In this example, the desktop layout has enough room to display three columns of page content. In the tablet layout, the mediaObjects and objectDescript with objectData div containers are displayed as two columns. And in the mobile layout, each div container appears on its own row to fit the narrow width of a smartphone screen. With the desktop layout mediaObjects and objectDescript with objectData div containers are displayed as three columns, see Figure 16.  
  


Figure 16. The desktop fluid grid layout.

At this point, all three layouts are defined. It's time to attach an external CSS style sheet to the index.html page to format the HTML that exists in each div container.

**Note:** Although you can add formatting styles to the CSS rules generated by the Fluid Grid Layout feature in Dreamweaver, this is not recommended. If you edit the rules that define the layouts (in mobile-web.css), you may lose the ability to edit the layouts on the grid. To maintain full editability, create your styles in a separate CSS file. This strategy also makes it easier to update the site later, since the formatting CSS rules and layout CSS rules are separate.

1. At the bottom of the CSS Styles panel, click the Attach Style Sheet button (see Figure 11). In the dialog box that appears, browse to select the file named formatting.css located in the css subfolder of the Web\_Mobile\_App folder. Leave all the other default settings and click OK.



Figure 17. Attach the provided external style sheet to format the site content.

1. Choose File > Save All, and preview in a browser. See Figure 18, which illustrates the tablet layout.  
     
     
     
   Figure 18. Initial format styles applied.

## Part 2 — CSS Style the fluid grid layout

In Part 1 we developed the developed a fluid grid layout for a Web Mobile App based on an Adobe tutorial developed by Tommi West, see Figure 18, and our modifications were colour coded in green text. The remaining parts of these tutorials are written specifically for this workshop, so we will dispense with the green text.

At this point, our results from Part 1 require several improvements, the first of which is to apply a better CSS style to our fluid grid layout. As mentioned above, we would like to keep separate our styling CSS (format.css) from our fluid grid CSS (mobile\_web.css) so that we don’t corrupt the fluid grid layout by accident. Of course, if you are careful you could combine the two. We will keep it separate, and you will also see later that we rely on other CSS style sheets already created for the various technologies we use. Because of this we also have to be careful where we call these style sheets in the Web3D Mobile App index.html file.

Before we start, let’ take a look at the completed\_part\_2 index.html file.

1. In Dreamweaver go to the files panel and open the index.html file in the completed\_part\_2 folder. Then click on the Preview/Debug in Browser button and select Preview in Google Chrome, see Figure 19.

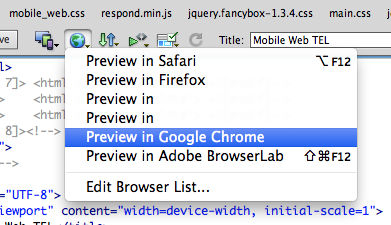
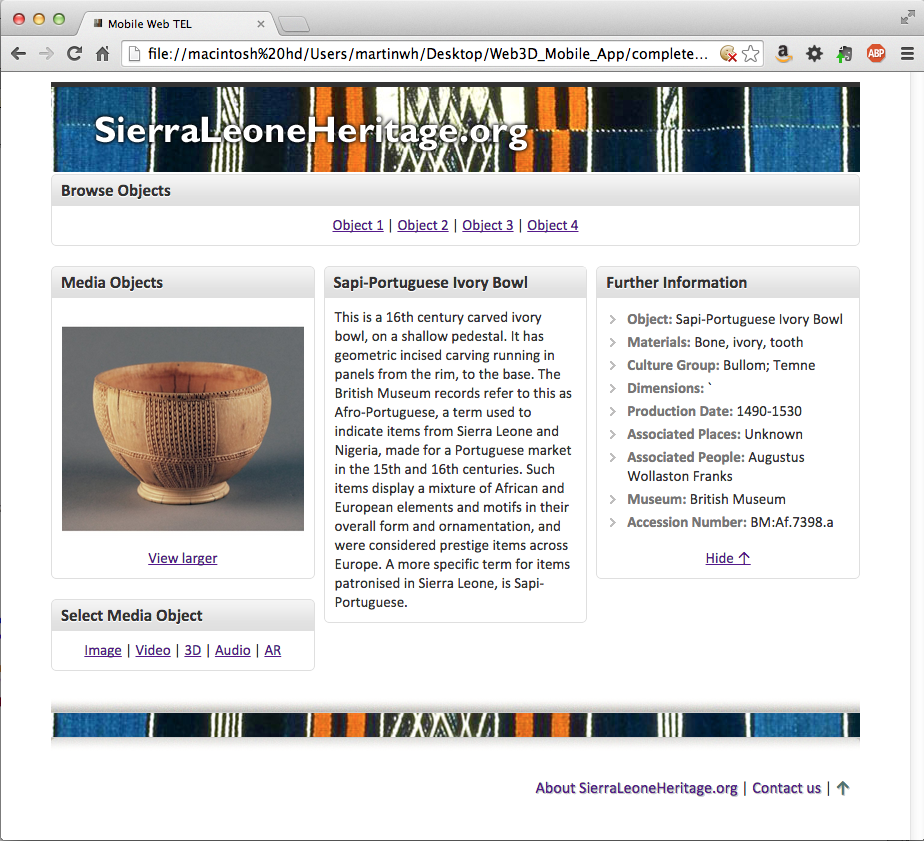
  
  
Figure 19. Select the Preview in Google Chrome option.  
  
You should now be able to browse the finished styled Web3D Mobile App, which you created in Part 1, see Figure 20.  
  


Figure 20. Part 1 results now styled largely with the Sierra Leone main.css style rules applied.

Have a go at resizing the app between browser, smartphone and tablet screen sizes. As you can see your part 1 results are now styled and responsive, let’s see how we did it!

You may recall that the data we have used so far is that already available from the Sierra Leone Heritage repository. As well as using data from this source, we will utilize the CSS style sheets belonging to this repository and web site. These are quite complicated, and a discussion of the details concerning CSS styling are beyond the scope of this workshop, so at this stage we will assume the ‘magical CSS styling fairy’ has arrived and created for you the Sierra Leone Heritage Web Mobile App main.css styles.

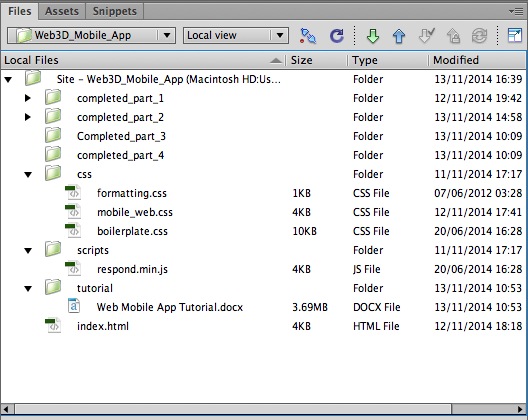
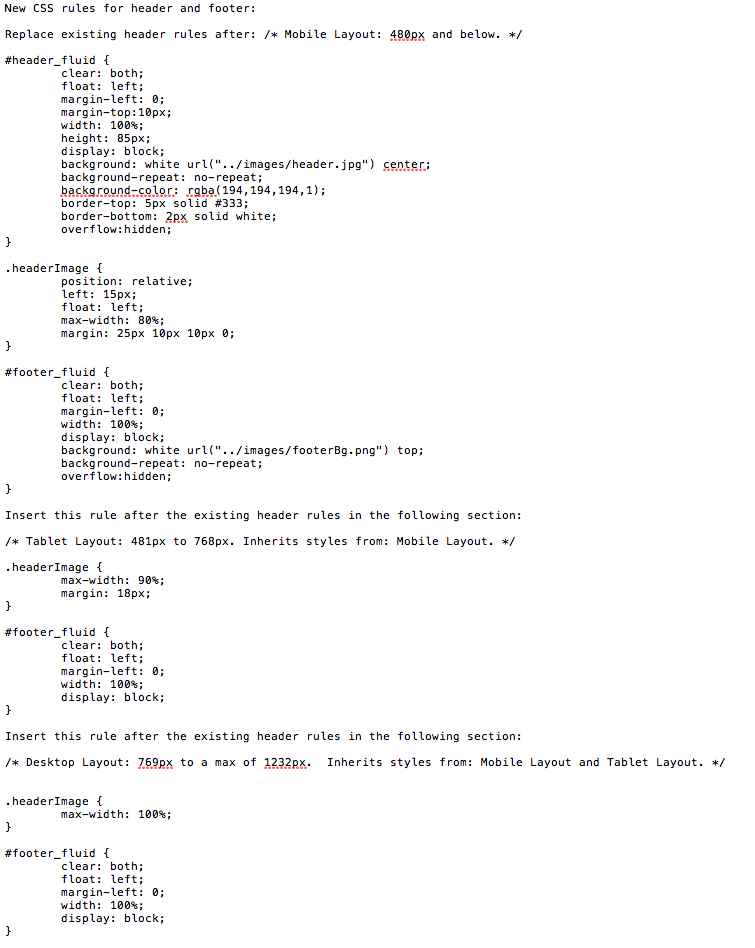
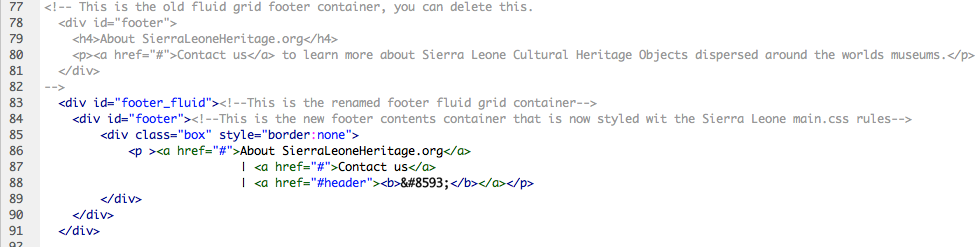
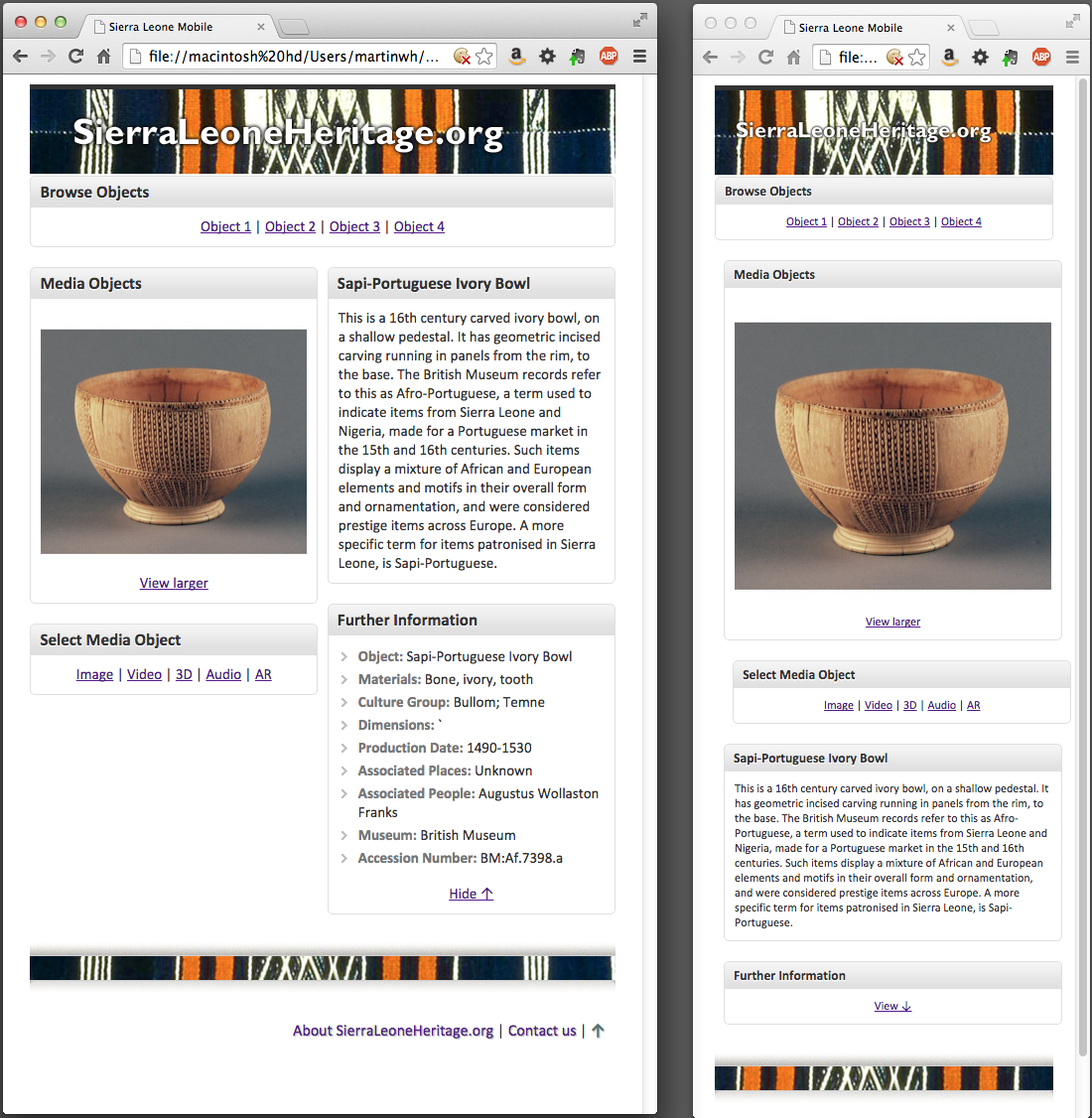
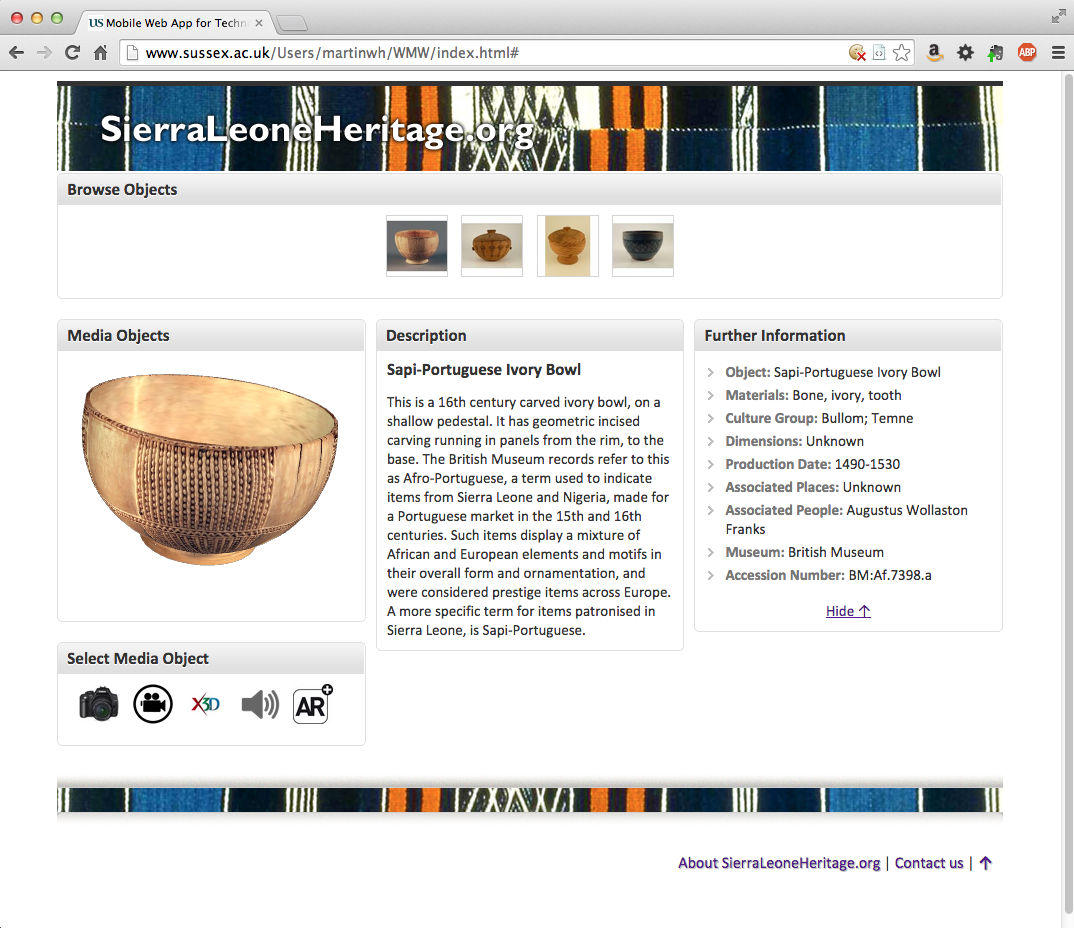
1. So, close the browser showing the Web3D Mobile App in Figure 20, and close the index.html file you have open in Dreamweaver (this is the completed\_part\_2/index.html version.
2. Now open the index.html version from Part 1 that you created. We will edit this and its associated files, e.g. mobile\_web.css. In actual fact, in Part 2 you won’t use the formatting.css file, so you will need to delete reference to this in the index.html file. We’ll get to that in a minute.
3. Let’s just check that you have your local files looking something like that shown in Figure 21.  
     
     
     
   Figure 21. The local file structure so far.  
     
   You should be able to see the 4 completed parts – these are for ‘here’s one we made earlier’! You should have your index.html from Part 1 with its associated css, scripts and tutorial folder.
4. Now open the completed\_part\_2 folder followed by its content folder and open (double click) the content\_part\_2.txt folder. Like in Part 1 you will cut and paste contents from here into your index.html file to replace existing contents. You might want to open content\_part\_2.txt in another text editor, e.g. Notepad++ if it is easier, and resize Dreamweaver and your text editor to appear side by side to make editing more efficient.
5. In the content\_part\_2.txt you will see 4 main sections of contents that will be used to replace contents in your index.html file:
   * **Content for the head section**. This content links a fancybox.css style sheet (open source) to allow the cultural object image to be enlarged in a light box, we may or may not use this later as a feature of the Web3D Mobile App. We also link to the Sierra Leone main.css style sheet. As mentioned before going into CSS in detail is beyond the scope of this workshop, so for efficiency we are using open source and already prepared CSS where we can to illustrate. One thing to note, however, is that the order of CSS style sheets linked in your HTML file can impact the styling of elements. For example, a later CSS rule may overwrite a previous one.  
       
     We also need access to some JQuery functions already available in open source libraries (jquery.js and fancybox.js) and the Sierra Leone main.js library.  
       
     You should now cut and paste the 5 lines of code (2 <link …> and 3 <script …> immediately below the <script src="scripts/respond.min.js"></script> around line 28 in Dreamweaver, see Figure 22. Save your index.html file  
       
       
       
     Figure 22. Inserting the CSS and JQuery files
   * **New content for header.** At this point, if you took a sneak preview at the index.html file you will see that the main.css rules have been applied to various elements of the HTML. It doesn’t look too pretty, but bare with it. Those guys who wrote the main.css didn’t do too bad a job! One issue that has arisen is that the main.css rules were originally designed for a non-responsive web site, i.e. it does not accommodate a fluid grid layout. The main.css file has rules for styling the header and footer containers and these are overwriting the fluid grid css style for the header and footer in the mobile\_web.css rules. There are a number of ways to get round this: 1) Open the main.css file and modify it to remove its styles fort eh header and footer, but we don’t want to do this to a third party CSS resource, 2) we could change the id=”header” to id=”header\_fluid” in our mobile\_web.css file, and I am sure there are other solutions.  
       
     We’ll use method 2). Cut and paste the new content for header code into index.html to replace the existing see Figure 23.  
       
       
       
     Figure 23. The new header container insert into the HTML  
       
     You will also need to update the CSS rules in mobile\_web.css for #header. Open up mobile\_web.css and replace with the new rules shown in Figure 24 and can be found in content\_part\_2.txt file. Basically, these css rules insert a background image behind the Sierra Leone logo in the header container, and also position and scale the Sierra Leone logo across the three device screen sizes.  
       
       
     Figure 23. This is the new css rules for styling the header (and footer) containers.
   * **New content for footer.** Similarly, we have to replace the existing footer container, but we’ll take a slightly different approach because we want to use the main.css styling for the footer contents. In this case we will rename the existing footer container id to footer\_fluid and update the mobile\_web.css. The new rules for the footer\_fluid rules are also show in Figure 23 and can be found in content\_part\_2.txt. Edit the mobile\_wb.css file appropriately.  
     You will also need to update the HTML footer code in index.html. The new code can be found in contents\_part\_2.txt at the bottom of the file. See Figure 24. Notice here we have effectively wrapped a new footer contents container: <div id="footer”>, which is styled by main.css, inside the renamed fluid grid <div id="footer\_fluid"> container. See Figure 24.  
       
       
     Figure 24. The new footer containers inserted into the index.html file. The old footer container has been commented out, but could be deleted.
   * **New contents for the remaining container.** We need to modify the objectPageSelector, mediaObjects, objectMediaSelector, objectDescription and objectData containers to provide css style rules from the main.css file reference earlier. For most of this css styling we are going to make use of the main.css box class and the associated boxTitle class. You could edit each of the containers or simply replace all the containers and contents with the HTML in the contnets\_part\_2.txt file. Go ahead and do that. One last thing to do is to remove the old format.css rules. To do this, either comment out this line:   
       
     <link href="css/formatting.css" rel="stylesheet" type="text/css">  
       
     Alternatively, delete it. The old css rules in that file are no good to us now, because main.css has lots of rules that style all the basic HTML elements.
6. Preview your index.html file as before, it should look like that shown in Figure 20 in desktop size, while Figure 25 illustrates the mobile and tablet fluid grid layouts.  
     
   

Figure 25. Part 2 version of the Web3D Mobile App Sierra Leon styling applied

Ok, that completes Part 2 CSS Style the fluid grid layout. Now for the really hard bit, we need to make this Web3D Mobile App dynamic and add some interesting media contents, not just an image and text! In **Part 3** **Add dynamic update to the styled responsive fluid grid layout** use a similar approach to swapping out existing HTML code for new code, and adding new JQuery code to perfom the dynamic updates. We will also build a few new css rules, with which we will create a new format.css file.

## Part 3 — Add dynamic update to the styled responsive fluid grid layout

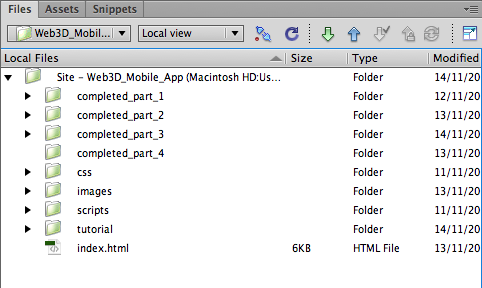
In Part 3 of this tutorial we will we need to connect up the various links (Browse Objects and Select Media Object links) shown in Figure 25, which select different cultural objects and the media objects associated with them. During this workshop you will have had a chance to create a 3D object and seen demos on how these could be used in Web3D and AR scenarios. A key requirement is to make the Web Mobile App dynamic so that it loads cultural object data and associated media (e.g. the 3D objects, video, etc.) from a data source, e.g. a database or JSON file.

Before we start let’s have a look at a finished Part 3 version, see Figure 26.  
  
  
  
Figure 26. The Web3D Mobile App showing cultural object 1 selected with the 3D model being animated.

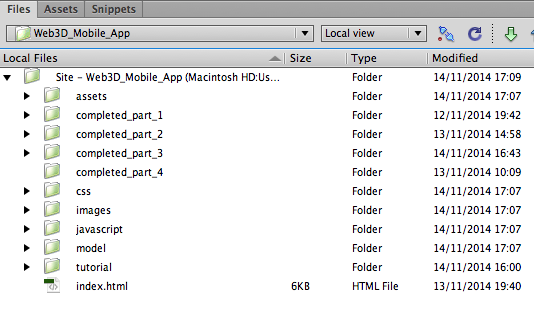
This final version can be accessed at [http://www.sussex.ac.uk/Users/martinwh/WMW/index.html#](http://www.sussex.ac.uk/Users/martinwh/WMW/index.html), open a web browser on the desktop, tablet or smartphone and have a look. It has many interesting features:

* + It is responsive across different devices: smartphone, tablet and desktop
  + It dynamical loads objects from a data source, limited to 4 at the moment, but you can easily add a defined number of objects with little code modification.
* It could be made more sophisticated to perform a search of objects on a data source and dynamically build the Browse Objects function, perhaps as a gallery. See the discussion at the start of this tutorial for Part 5.
  + It utilizes JQuery (a popular JavaScript API) to add interaction to the application
  + It integrates a wide range of media objects (images, 3D, video, audio, AR trigger images) for use in a teaching and learning scenario based on 3D and AR, e.g. the Magic Book and online 3D
  + It can be adapted to other simple scenarios, data sources and styling
  + …

Ok, let’s start to create the dynamic version illustrated in Figure 26.

1. You should still have Dreamweaver open, and your local file structure should look something like that shown in Figure 27.  
     
     
     
   Figure 27. The local file structure so far.  
     
   If you look inside the completed\_part\_3 folder you will find all the files needed to create the Web3D Mobile App illustrated in Figure 26. Whereas the css, images and scripts folder at the Site – Web3D\_Mobile\_App level belong to Part 2, which you just finished. Note: a completed copy of Part 2 also resides in the completed\_part\_2 folder. Likewise, the index.html folder you see in Figure 27 is the Part 2 version you just completed.

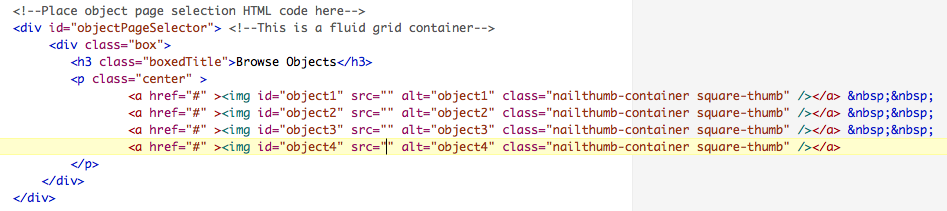
These are your Part 2 files

1. If it is not already open, open the index.html version from Part 2 that you created. We will continue to edit this and its associated files, located in several new folders. In Part 2 you will need more code files (that we previously created for this tutorial), which use JavaScript and JQuery to provide interaction and dynamic update of cultural object, and you will apply some more css rules, which are now located in the formatting.css file – remember we try to keep fluid grid css (mobile-web.css) aprt from general formatting css (formatting.css). This is to try and ensure we don’t accidentally corrupt the fluid grid layout.   
     
   You will also need a data store and some media assets (lot’s of media assets, with only 4 cultural objects illustrated the media objects assets are growing rapidly; 20 in total, too many to create in this workshop).  
     
   So the first thing we will do is copy over some folders/files from the completed\_part\_3 folder before we begin to update the index.html file. These folders will contain code that is beyond the scope of this workshop, but necessary for the Web3D Mobile App to function. Having said that, surprisingly little JQuery code has been written from scratch. The app relies mostly on third party existing JQuery libraries and CSS.   
     
   Another interesting point is that JavaScript libraries or APIs such as JQuery abstract away a lot of the code and issues that you would have to deal with if you coded the Web3D Mobile App from scratch in just JavaScript. For example, each web browser handles HTML is slightly different ways, meaning you would have to accommodate this in your code. JQuery, handles this for you, ensuring your responsive web application runs in all common web browsers.  
     
   At this stage, because there are a lot of new files in various folders created to make Part 3 work, it will be simpler to overwrite all the folders in your Part 2 folder (Site – Web3D Mobile App) — don’t worry, there is a copy in the completed\_part\_2 folder.
2. So, copy all the folders in completed\_part\_3 folder into the same folder as your Part 2 index.html file. You will see that Dreamweaver will ask to update and replace existing files, etc. Reply Yes to All. However, you will need to manually delete the Part 2 scripts folder, because the new folder that contains your javascript files is now called javascripts, so the scripts folder doesn’t get overwritten. Make sure you leave the Part 2 index.html file as it is.  
     
   We will limit this stage (Part 3) of the tutorial to a brief discussion of what each new file does as we update the Part 2 index.html file to create the Part 3 index.html file. If all goes well, you will then copy the Part 3 Web3D mobile App to the University web server to see if it works, fingers crossed!  
     
   Your local files folder should now look like that shown in Figure 28.  
     
     
     
   Figure 28. Your local files set up to start Part 3, where we will modify the index.html file to make it dynamic, i.e. load cultural objects from an external data source.
3. Next check that the Part 2 index.html file still works with all the new files you just setup (installed in the Part 2 Site – Web3D Mobile App folder where your index.html resides). It should do, because buried within those files in the css, javascript, images, etc. folders is still the original css rules, and fluid grid css and javascript code created in Part 1 and part 2 of this tutorial. Check by previewing in a browser as you did in Part 1 and Part 2 of this tutorial.  
     
   As we go through updating and modifying the Part 2 index.html code to make it dynamic we will have to test the results in a web server — it will no longer be feasible to preview as a file, because the whole point of dynamic iis to update the application with new data from a server based on an HTTP request. To do this we have arranged test accounts fort his workshop on the University web server.   
     
   Another way of testing your dynamic Part 3 version is to use a local host, which is a server installed locally on your Mac or PC. Macs actually have web server built into the operating system, or you can down load something like MAMP. For a PC you would download something like WampServer, but there are many variants to choose from that allow local development and testing.

However, for this tutorial, all you need to do is copy your complete Web3D\_Mobile\_App subfolders to the public\_html folder at the root level on your **Home Directory (N:)** drive. We’ll show you how to do this as we move forward (if we get this far).  
  
The next thing we will do is one by one replace each of the 5 inner containers (these are the objectPageSelector, mediaObjects, objectMediaSelector, objectDescription and objectData containers), because each of these need to access contents dynamically through an AJAX (Asynchronous JavaScript and XML) request to an external data source that holds the associated data. For example:

* The **objectPageSelector** container needs to access an image (from the data store) for each cultural object and use this as a thumbnail to select the current cultural object.
* The **mediaObjects** container needs to access the actual media object that is selected for display that is associated with the cultural object. It is common for such media objects to be store in a media sever that is referenced by a URL, which itself is stored in the external data store. Examples of media objects we will be using are: images, video, 3D, audio and trigger images for the AR (Augmented Reality) Magic Book.
* The **objectMediaSelector** needs to access icon images, which again could be crated from the original cultural object image referenced in the external data store, but in this case we have sourced some nice looking open source icons from the InterWseb and stored them in an asset/icons folder. Again, such images might also be accessed from a media server.
* The **objectDescription** container needs to access the title and textual description of the cultural object (selected by the objectPageSelector) from an external data source.
* The **objectData** container needs to access the cultural object’s (selected by the objectPageSelector) metadata (or Further Information) from an external data source.

Once you have replaced these 5 containers with new versions, don’t be alarmed that your Web3D App no longer works, you still have to integrate the appropriate JQuery and css files that make it dynamic.

1. We might as well start with the objectPageSelector container. Open the content\_part\_3.txt file, copy the new content for the objectPageSelector container illustrated in Figure 29. Use this code to replace the existing objectPageSelector in index.html  
     
     
     
   Figure 29. The new objectPageSelector container code, found in content\_part\_3.txt.

If we look at this HTML code for a second, what is it that makes it dynamic? As mentioned above, accessing an image (from the data store) for each cultural object and use this as a thumbnail to select the current cultural object. To access this image we use a JQuery function to read the JSON file that contains the image. The JQuery function is located in the getCulturalObjects.js file in the javascripts folder. A specific piece of code in the getCulturalObjects.js is responsible for retrieving the imamge file that will be processed into a thumbnail, Figure 30 illustrates, and you can examine the JavaScript files for more details – explanations are largely covered in the code comments for convenience.

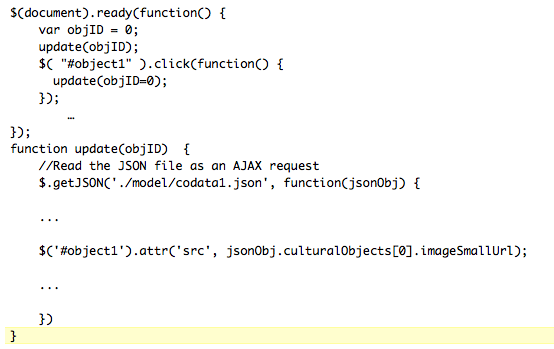


Figure 30. JQuery code snippet illustrates how the JSON file (coData1.json) is read to retrieve data.

In this case a small image is retrieved from the data store that will be used to create a thumbnail. So how do we assign this image to the appropriate part of the HTML code. This is done with the ID #object1 tag. If you go back and look at the HTML code for the objectPageSelector container you will see the same ID #object1 tag. This tells JQuery tp place the image in this <img></img> container, and of course it places it between the quotation marks in the src attribute part of the <img> tag. What is actually placed here is the URL of the image that has been retrieved by the $('#object1').attr('src', jsonObj.culturalObjects[0].imageSmallUrl); function.

Otherthings are happening here too, we mentioned the thumbnail is created. This is done be invoking another JQuery function (via the class="nailthumb-container square-thumb") located in the createThumbnails.js file. This simply takes the image returned by the src url and turns it into a thumbnail image before inserting the thumbnail in the <a href=”#”> *image goes here*</a> tag.  
  
Note, id and class are used to identify elements that will be styled in css or have contents replaced. The difference between an id and a class in HTML5 is that id’s can only be used once, so you have to have a unique id each time, where as a class can be used several times, i.e. in other elements. In the current example, we have unique objects, so each object is identified with its unique id, but we need a generic function to create thumbnails, hence the thumbnail class.

1. Next repeat step 5 to replace the remaining containers: mediaObjects, objectMediaSelector, objectDecription and objectData. We could go through each of these containers and describe then in detail, but this tutorial would grow exponentially. The principle is the same, as described briefly in Step 4 above, each container access dynamic dat. This dynamic data, e.g., the media objects, are is accessed by the getCulturalObjects.js and located into the appropriate HTML elements via the associated id or class.  
     
   It is interesting to look at the new mediaObjects container, notice we no longer have any hardwired data, same for the other containers. Each media object is associated with a class: objImage, objVideo, obj3D (this is even more interesting, it uses a technology called X3DOM, basically a 3D object formatted in XML that takes advantage of the x3dom.js file to render the 3D in the HTML5 DOM), objSound and obArx3d (this simply identifies an AR trigger image).
2. Finally, you now need to link in the associated css and javascript file in the head of the index.html file. If you go to the top of the content\_part\_3.txt file you will see a large number of links and scripts. Some of these are already in your index.html file, but it may be easier to just replace the existing with these. Note. These are also heavily commented with explanations as to what they do. Go ahead and integrate this new content for the head section. but you must ensure that the css links and javascript files are insert in the order given in contents\_part\_3.txt. This is because of conflicting css rules over writing other rules.   
     
   It may also be interesting to insert each group of css links and javascript files on by one testing to see the effect.

Finally, as mentioned above in Step 4, to test this Part 3 version all you need to do is copy your complete Web3D\_Mobile\_App subfolders (for Part 3) to the public\_html folder at the root level on your **Home Directory (N:)** drive. If you get stuck here, we’ll give you a hand.

Some last words, when we envisaged this work 3 weeks ago, we hadn’t actually built the Web3D Mobile App, and we hadn’t created any contents. On hindsight, it would have been better to run it over a whole day at least, so we could get more into finishing the Web3D Mobile App. The envisaged Part 4 and part 5 are what will turn it into something really useful, but we think you get the picture.

Hope you learned something useful, time to wrap up!

# Part 4 — Add more objects to the Web3D Mobile App data store.

To follow.

# Part 5 — Connect the Web3D Mobile App to the Sierra Leone Heritage repository.

To follow.