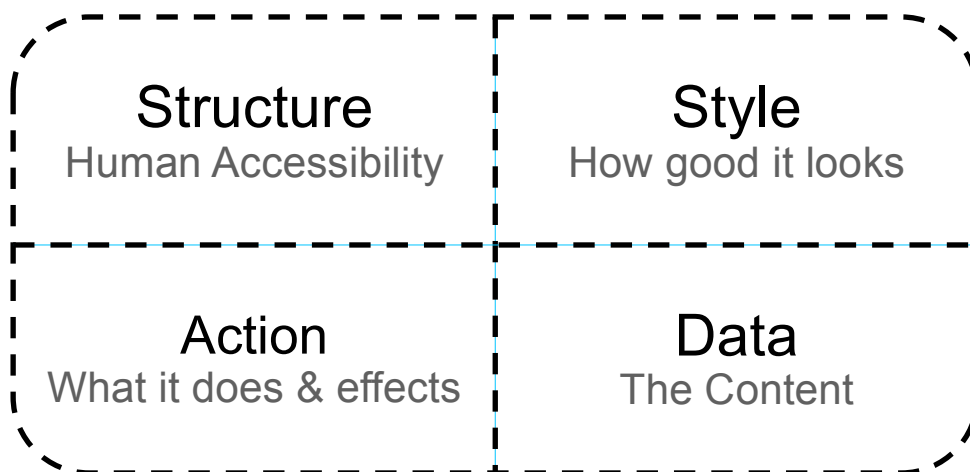


Web Building Blocks

In this exercise we are going to look at the four building blocks of the web, structure, style, content and action. This methodology forms the basis of a data driven web site. Additionally by moving data to be a building block of the web, the same methodology can be used to expose data, making it of the web, not on the web.

In this exercise we shall be using jsFiddle.net to build a simple data driven web site in HTML5.

Building Blocks



Structure: The structure of a web page. Designed to enable human consumption of the content no matter the type of user or device. Accessibility is key and most pages misuse structure in order to add style, making content very hard to access for disabled and visually impaired. When was the last time you accessed your web site using a screen reader?

Style: How the web page looks. Including where the elements are on the page, what colours and borders things have and what is hidden from view. Using style sheets you can define multiple presentations of your content for different media and users, e.g. desktop, mobile and print mediums.

Action: What the page does. Using action you can define interaction with the content. Simple examples include dynamically loading content based on user choices, controlling embedded video and multimedia, and rotating banner news articles to promote content.

Data: Your content. Data is machine readable, e.g. in a database or provided by a data provider, e.g. calendar and news/blog feed system. Data is the content which gets embedded in your web site for presentation to a user. Traditionally web servers obtain the data from an internal system and ONLY present it on a web page, designed only for human consumption. Equally, you can serialise your data into other formats in addition to your HTML and expose the data in a machine readable fashion.

Exercise

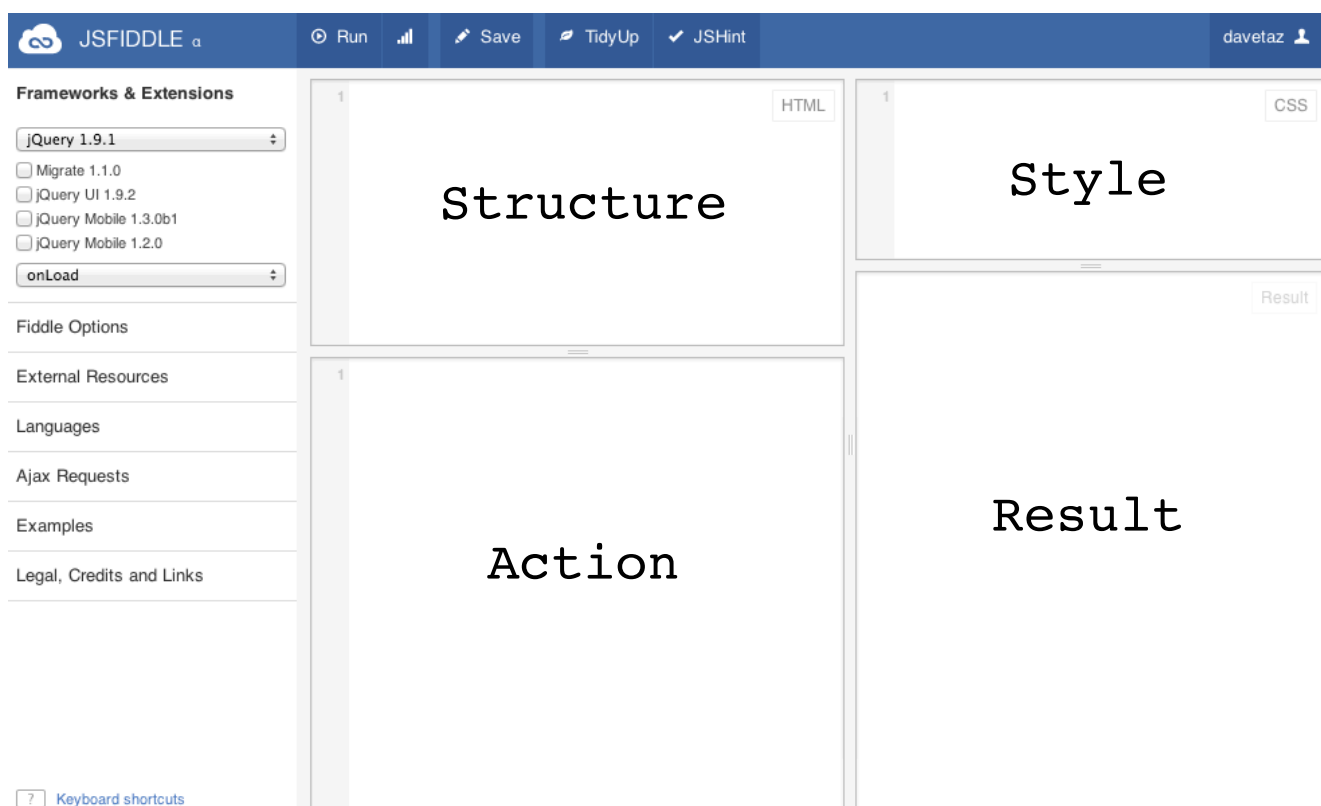
The idea of this exercise is to introduce you to HTML5 and how to separate structure, style, action and data. To do this we are going to create a simple web page with a heading and summary that is loaded in from our data. As a more advanced exercise, we shall expand our page to dynamically load in a news feed from a third party website.

In addition to designing a simple web site, we shall also look at how you can expose the raw data and demonstrate how this can be done in one line of code.

jsFiddle.net

jsFiddle (shown below) is an online sandbox which allows you freely to experiment with the building blocks of the web. It has three main panels that allow you to define structure, style and action related to your web page. The fourth panel (bottom right) shows the result of your editing; this panel is updated by pressing the run button at the top of the screen.

For the purposes of this exercise, please **select jQuery 1.9.1** from the “Frameworks & Extensions” panel on the left hand side as shown.



As jsFiddle does not have a section where you can define data directly, we shall be using the action section to build our objects in JavaScript, the language of action on the web.

1. Structure

We are going to start in the structure block and define the content we are going to put in our page. For this we are going to use three HTML5 elements:

- <header>** A page or site header block. Normally used across the entire site.
- <summary>** Summarises the content of the page and why users would want to read further.
- <section>** A section on your site, in this case it will contain our news

To make our structure then we simply need to define the following in our structure block:

```
<header>Page Title</header>
<summary>Page Summary</summary>
<section id="news"></section>
<data></data>
```

As with any markup language, we are creating tags which we open “<tag>” and close “</tag>”. For the purposes of clarity we have added some text “Page Title” and “Page Summary” which will be displayed on our web page.

As a web page is likely to have many sections, we have assigned an **id** attribute to our section to say that it will display our news.

Once done, if you press **run** you should see your page displaying a title and summary sections with the text you have defined. Congratulations if this is your first HTML5 page.

1.1 Completing the HTML5 structure

This part is only relevant when you are implementing an HTML5 page natively and not in JSFiddle where the extra bits are added for you automatically.

Although this looks like an HTML5 page, JSFiddle is adding many of the required HTML5 tags in order to make this example work. A fuller HTML5 example follows. Note that in this example we have used indentation to show where tags begin and end.

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Page Title</title>
  </head>
  <body>
    <header>Page Title</header>
    <summary>Page Summary</summary>
    <section id="news"></section>
    <data></data>
  </body>
</html>
```

If you would like to check this HTML5 is valid why not try the validator at <http://validator.w3.org/>

2. Data

In order to create some data we are going to use the action block and define our data as a simple JavaScript object. Normally you might load data from a database or other external source, but at some point you will be handling objects or records, so we may as well cover these here.

In order to create some data we are simply going to create an object and give it some properties defining the *header* and *summary* of the page we are on. This way the data is not part of the structure, but actually data.

In the action block, define the following:

```
// Create our data object so we can put things
in it.
var data = {};

// Define a header and summary
data.header = "BBC News Aggregator";
data.summary = "This page lists the top 7
articles on the BBC News Feed";
```

Feel free to change the content in quotes to whatever you want, at this point we have a data structure.

Before we continue we are going to serialise this data in the JSON (JavaScript Object Notation) format. This is a data serialisation common on the web that developers like to use, as it is so easy to integrate with services. So easy in fact, you have just built a service that can read JSON objects!

Our object *data* is already a JSON object, however to share it openly we need to serialise it. Serialise is basically a flash way of saying “Turn it into a chunk of structured text” known as a string.

```
// Get a json serialisation of our data into a
variable called json_string

var json_string = JSON.stringify(data);
```

In order to output this object for people to use we are going to put it inside our `<data>` node. This is done by simply setting the content of the data node to the *json_string*. We are going to do this using jQuery notation:

```
// Output the raw data into the data node

$('data').html(json_string);
```

Beware, although it looks like we have output the JSON data into our web page, what we have actually done is serialised our *data* object as a JSON string, and then serialised this string as HTML. This operation will result in any non-standard HTML characters being translated into HTML. This means although it looks like a JSON object to a human, to a computer it is an HTML object. To correct this we should output a raw JSON file from our web server, something that JSFiddle is not able to do for us.

At this point feel free to copy the outputted data object (from the first “{” to the last “}” inclusive) and paste this into <http://jsonlint.com/> in order to validate this object as valid JSON.

2.1 Challenge One

Up to this point I've been doing all the work for you. Your challenge now is to replace the content of your *header* and *summary* tags with the data from our object. When done, write your working solution in the box below. Ideally it should be no more than two lines long.



3. Style

So far we have built an accessible web page that provides data to users. At this point the data is more accessible to machines than humans, who care about how things look! So let's add some basic style.

Style is defined using the CSS (Cascading Style Sheet) language, the latest version being CSS3. The examples presented here are very simple and there are many good tutorials available online.

Firstly let's style the header block:

```
header {  
  font-size: 2em;  
  color: white;  
  text-align: center;  
  background: #900;  
  padding: 0.2em;  
}
```

Feel free to have a play with these values to see what changes occur.

3.1 Challenge Two

Add some style to your summary block without touching the structure to make it more appealing as a subtext. Maybe you might want to look up how to do italics in CSS3.

4. Dynamic Content with RSS (The Advanced Section)

In this section we are going to look at using Asynchronous Javascript And XML (AJAX) request to dynamically load and update data from a third party website, in this case BBC News Worldwide.

First thing to note is that while AJAX as a technology evolved as what it stands for, people are now abusing it for doing both non-XML requests (as we are about to do) and also synchronous requests. The only bits left really are the fact that AJAX is a technique in JavaScript to load resources from a remote location.

First we need to create a new piece of data containing the URL of our news feed.

```
data.url = "http://users.ecs.soton.ac.uk/dt2/odi/news/";
```

We can then create a function in our action section (at the bottom), that loads our news from this URL.

```
function load_news(url, json) {
  // Begin ajax request
  $.ajax({
    type: "GET",
    async:false,
    // Load the requested url
    url: url,
    // use jsonp as data type as we are doing cross-domain
    requests, security reasons.
    dataType: 'jsonp',
    timeout: 5000,
    // If successful we will get data back.
    success: function (data) {
      var html = '';
      // Iterate over each "item" in the data
      $.each(data.stories, function (index, item) {
        // Turn each article into an HTML5 article node
        html += "<article>" + item.title + "</article>";
      });
      if (!json) {
        // If we didn't want the raw json back then output
        the html
        $('#news').html(html);
      } else {
        // If we want to display the raw data, output this
        instead.
        $('#news').html(JSON.stringify(data));
      }
    },
    // If there was an error, tell the user
    error: function (error) {
      $('#news').html("Error, Unable to Load Content");
    }
  });
}
```

In this code section, data is being both processed and displayed in the *news* section of our web page. Significantly as each item of news is processed, each is contained within an `<article>` HTML5 tag, meaning that now our action section is also adding structure to our document. This is a perfectly valid

use in this case, as within the section titled news we would expect to find a number of articles; this makes screen readers very happy.

Note that in order to make this function work, something needs to invoke (or call) it. To do this simply add the following line to your action section somewhere underneath your data declarations.

```
load_news(data.url, false);
```

4.1 Extension exercises

These are the exercises for those feeling very brave.

- How about outputting the raw data about the news (hint: what does changing *false* to *true* in the above do?)
- Try adding style to your news articles

```
article:before {  
    content:"\2022 ";  
}
```

- Try adding a button you can press to update the news content on the page.

```
$('#button').click(function() {  
    load_news(data.url, false);  
});
```

- How about adding URLs to the news articles so people can navigate to them

```
item.link
```

- Change the update button to be a countdown timer that loads the next update after x seconds.

```
setInterval(function() {  
    load_news(data.url, false);  
},3000);
```

- Add another section and button to show the raw data.
- Highlight all instances of a keyword in the news.



<http://jsfiddle.net/davetaz/zHPNK>