Lab 3: Using an RSS Feed

Objectives of this lab session:

- Use the jGFeed plug-in with an app to get access to RSS feeds
- Examine the RSS data returned by jGFeed in the Chrome debugger

Resources required:

- A current web browser (ideally Google Chrome)
- WebStorm IDE
- jQuery & jQuery Mobile libraries (a link to these is fine unless you wish to work offline)
- The jGFeed plug in code (get a copy of jgfeed.js from the jGFeedDemo app in the Moodle Examples and Demos section)

Part 1: Create a Simple Feed App

In this stage, you'll create a simple feed project to test jGFeed:

1. Start a new project (RSS-Reader) and add a HTML file to it (rss-reader.html is a suitable name).
2. Link in the jQuery and jQuery Mobile libraries and add the **<meta>** viewport element as usual (see labs 1 & 2)
3. Find the **jgfeed.js** file by a Google search or from the jGFeedDemo app on Moodle; copy this file into the new project's folder. Add a link to the file – note, this must come after the jQuery and jQuery Mobile links
4. Add a new JS file to the project (call it **rss-reader.js**) and add a link to it, after the link to **jgfeed.js**
5. Add a jQM page-div with the usual header, content and footer element-divs (see lab1)
6. The content div of the page will be a listview with (currently) no list items in it:

```
<ul id="list" data-role="listview">
</ul>
```

   Note the id (**"list"**) that we will use to update the list in Javascript. You should also add an attribute – **id="title"** to a heading tag in the page header

At this stage we are ready to add code to access an RSS feed and display it. Any RSS feed will do in this part, but so that you can follow the steps for identifying feed information, we'll go with a feed from the iTunes store to download information about current sales.

Part 2: Incorporating the Data Feed

Apple provides a number of RSS feeds associated with the iTunes store (i.e. they are all there to sell you stuff). However, they are simple to use and you can configure feeds with virtually no effort. Go to https://www.apple.com/rss/ to try this out. For the sake of example, we'll use a feed for the 10 top selling movies from iTunes.

There are several parts to this – fortunately, they are all simple.

1. **Add the feed's URL to the project**. Apple provides a set or RSS feeds associated with the iTunes store **https://itunes.apple.com/gb/rss/topmovies/limit=10/xml.** It is easiest to put this URL into a variable:

```
var feedURL = "https://itunes.apple.com/gb/rss/topmovies/limit=10/xml";
```

Note that in some situations, you may need to insert values into the middle of a feed URL. For example, if I wanted to allow the user to choose mow many top-selling movies to return in the list, and in that case, I'd need to insert a number after **"limit="** in the feed URL shown. You can do this easily in Javascript using the **String.replace()** function:

var feedURL = "**https://itunes.apple.com/gb/rss/topmovies/limit=~~/xml**";

Note that I've replaced the number 10 with some text that is unlikely to appear anywhere else in the feed (~~). I can now replace this with any number I fancy:

var n = 12, url = feedURL.replace("~~", n.toString());

2. **A useful utility function** can do the bulk of the work in accessing the feed. We can also add a little code to check whether the browser is currently online, and retrieve the last feed accessed from localStorage if it was not:

```
function getFeed(url, success){
    if(window.navigator.onLine) {
        $.jGFeed(url, function(feeds) {
            // Check for errors
            if(!feeds){
                // there was an error
                return;
            } else {
                // No error – we've got the feed data…
                localStorage.setItem(url, JSON.stringify(feeds));
                success(feeds.title, feeds.entries);
            }
        });
    } else {
        // Get the fall-back...
        var feed = JSON.parse(localStorage.getItem(url));
        if(feed && feed.length > 0) {
            success(feed.title, feed.entries);
        }
    }
}
```

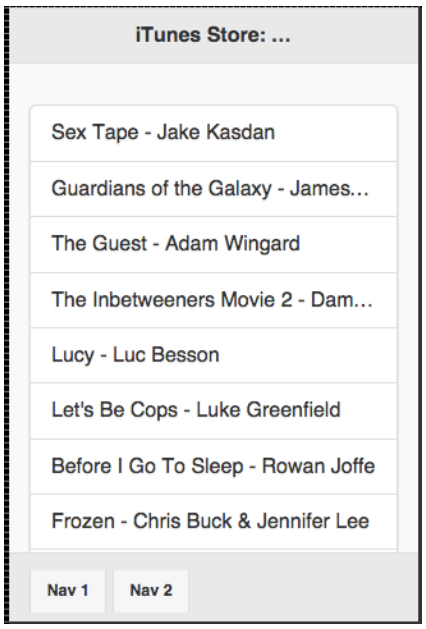
Note that if the device is offline, we get the fall-back result from localStorage. If the device is online and a feed is returned, the first thing we do is to stash the new data into localStorage, before calling the **success()** function to display the returned data.

3. **We can now add just enough code to get the feed data**. A sensible way to do this is to add code to the '**pagebeforeshow**' event, so that the list is updated before the page is displayed

```
$(document).on('pagebeforeshow', function(){
  getFeed( feedURL, function(title, items) {
    $("#title").text(title);
    var list = "", index;
    for(index=0; index<items.length; index+=1){
      list += "<li>" + items[index].title + "</li>"
    }
    $("#list").html(list).listview('refresh');
  });
});
```

The highlighted section is the "success" function that is passed to getFeed()

At this stage we should have a working app. **Test it by opening the page in a browser** (preferably Chrome because of the next stage) and (provided you've done all this correctly) you should see a page like the one shown in figure 1



Note – your results will be different from these since the data will be 'live' – i.e. films that were doing well on iTunes when this lab sheet was written.

Figure 1: Data returned from an RSS feed

Part 3: Digging deeper with the Chrome Developer tools

We can leave the app in this state and it will serve a useful purpose. However, with some knowledge of what goes on *inside* a feed we have a better chance of creating an app that is more distinctive and easier to use. The first stage is **to get some information about the raw data**:

1. With the RSS-Reader page in the Chrome browser, **open the developer tools**, **select the Sources tab and open the rss-reader.js file**. The function added to the code in step 3 above includes a call to the **getFeed()** function, which itself calls an anonymous function when feed data is returned from the feed's server. Place a breakpoint inside this function (on the first line, which starts **$("#title").text())**, return to the browser and refresh the page.
2. The code should stop at the breakpoint, and you now have access to the data passed to the function by an AJAX call made inside **jgFeed**. If you **hover your mouse pointer over the two parameters** to this function (title and items), you should be able to see their contents. Since the items parameter returns a whole tree of data, a better strategy is to use Chrome's Watch Expressions feature
3. **In the Watch Expressions page** at the top right of the Developer Tools, **press on the little '+' icon to add an expression**. Enter the name ***items*** and press the Enter key. You should be able to expand the items array to examine all of the properties returned by jgFeed. If any property is too long to view, remember you can open a console pane and enter it's name there (e.g. ***items[0].content***).

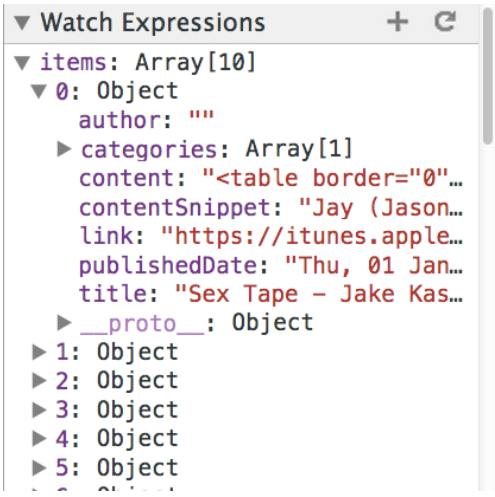


Figure 2: Feed response in Chrome developers' tools

Using this method, you can view details of any RSS feed as raw data. You can see that a feed is returned as an Object with a list of properties (**author**, **categories[]**, **content**, **contentSnippet**, **link**,

publishedDate and **title**). This is true of all RSS feeds, although there is some variation – for example, some feeds return a fairly small amount of content while the Apple RSS feeds tend to return a fairly complex chunk of HTML data in the **content** property. The **link property** is useful because that normally links to a full web page providing further information.

Formatting a feed

Now that we can access the details of a feed, it is quite easy to do some simple formatting to improve the look of the app. jQuery Mobile allows you to add mark-up inside each item in a listview, and that can include an icon or thumbnail graphic, headers, links etc. The easiest way to approach this is to provide a separate function to format an item from the list of feed items. We can do this as follows:

a) **Modify the start-up function** (**$(document).ready()**) so that it passes each item in the feed to a formatting function:

```
$(document).on("pagebeforeshow", function() {
    getFeed( feedURL, function(title, items) {
        $("#title").text(title);
        for(var index=0; index<items.length; index+=1){
            $("#list").append(formatItem(items[index]));
        }
        $("#list").listview('refresh');
    });
});
```

Note that our **list** variable is now built up by adding the return from the **formatItem()** function repeatedly.

b) **Create a formatting function** – for example:

```
function formatItem(item) {
    var li = document.createElement('li'),    // The main element returned
        a = document.createElement('a'),      // An <a> tag for the link
        p = document.createElement('p');      // The main text content
    a.setAttribute("href", item.link);
    a.setAttribute("style","white-space:normal;");
    a.innerText = item.title;
    p.innerText = item.contentSnippet;
    p.setAttribute("style", "white-space:normal;");
    a.appendChild(p);
    li.appendChild(a);
    return li;
}
```

Note that this function returns a **<li>** element, inside which is an anchor tag, so that the list item acts as a link to the linked web page. The content of the anchor tag is the item's **.title** property plus a **<p>** element containing the item's **.contentSnippet** property. The **<a>** and **<p>** tags have been given the style **"white-space:normal"**, which stops jQuery Mobile from truncating long lines with an ellipsis (…).

c) Test the app and you should see that each item is now formatted with a heading and the full content snippet, and that each list item has a **">"** symbol to its right, indicating that it will act as a link.

You can apply similar techniques to format any data returned by a RSS feed or any other source.

End of Lab 3