Building a multipage application

Overview

In this tutorial we will learn:

- The basics of a multi-page application
- How to load an external HTML file
- How to implement page navigation with history

The basics of a multi-page application

A MobileFirst hybrid application uses a single DOM model. A single DOM model means that you must never navigate between various HTML files by using hyperlinks or changing the `window.location` property.

Instead, you must implement **multi-page interfaces** by loading external HTML file **content**, and by using Ajax requests and injecting them into the existing DOM. This is required because the main application HTML file loads the MobileFirst client-side JavaScript™ framework files, and when the browser navigates from one HTML file to another, the JavaScript context and loaded scripts are lost.

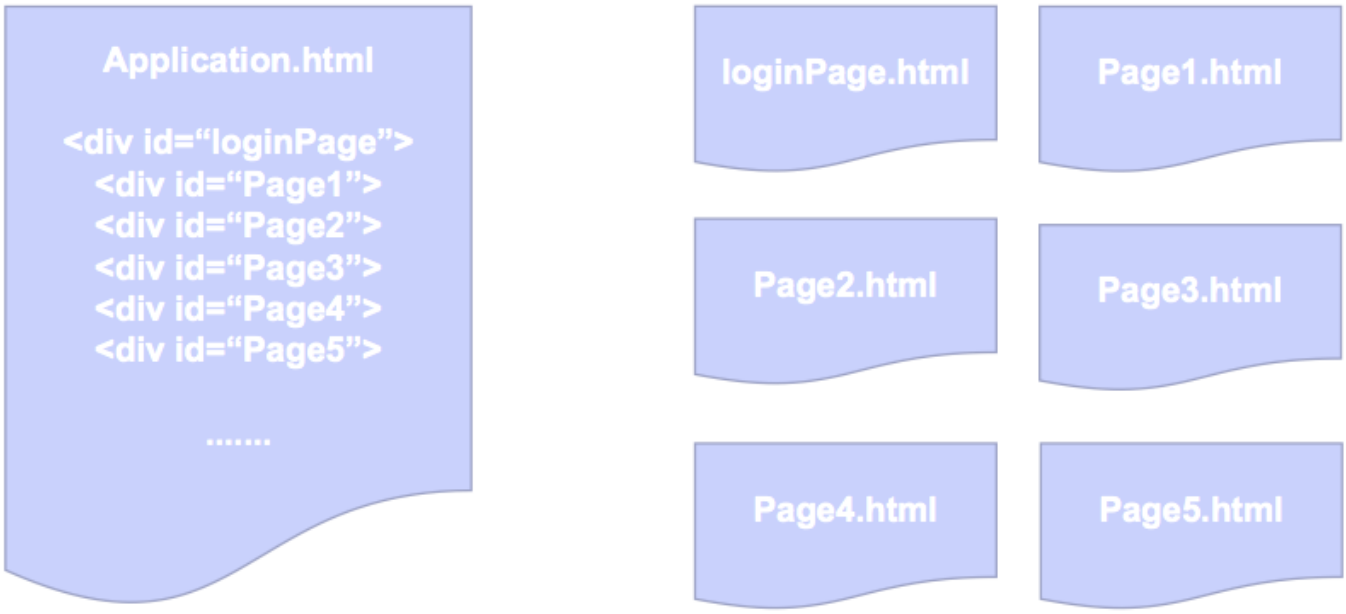
Most JavaScript UI frameworks that are available today (for example, jQuery Mobile and Dojo Mobile) provide extensive APIs to achieve the required multi-page navigation.

In this tutorial, a multi-page MobileFirst application implementation is explained by using **built-in functionality only** with jQuery.

Important: Do not use the built-in functionality that is described in this tuotiral if you intend on using a JavaScript UI framework. Use the framework's APIs instead.

Applications with multiple pages can be built in two ways:

1. Single HTML file that contains all the application "pages"
2. A separate HTML file for each application "page"



A single HTML file is the preferred model for simpler applications, where the developer is responsible for showing the "current" page `DIV` and hiding the rest.

However, larger applications present a challenge:

- The developer must take full responsibility for which DIVs are shown and which `DIV`s are hidden at any moment
- If new content is needed in a page, for example a table, a prepared template cannot be loaded, rather it must be generated manually
- A single large HTML file with many pages takes longer to load
- It is easy to get lost in a single HTML file that contains multiple "pages". Separate files are easier to manage

Building a rich dynamic application with multiple pages can be easier with dynamic page loading:

- Built-in jQuery APIs can be used to dynamically load, update, and insert DOM elements in your application
- HTML pages with CSS and JavaScript can be inserted as needed
- Navigation history can be implemented
- JavaScript code can be executed when pages are loaded or unloaded

Implementation notes:

- When you implement multi-page navigation in the Windows Phone 8 environment, you **must** change the URL each time you use the jQuery `load()` API method. *The example project for this training module demonstrates in detail how you can perform this task.*
  Add `/www/default/` at the beginning of the URL path string. For example: Change

  ```
  $("#pagePort").load("pages/MainPage.html", function(){
     (currentPage.init);
  });
  ```

  To

  ```
  $("#pagePort").load("/www/default/pages/MainPage.html", function(){
     (currentPage.init);
  });
  ```

- When implementing multipage navigation for the Windows Phone 8 environment, and jQuery Mobile is used with the `changePage()` API method, a jQuery Mobile defect prevents it from properly working. To overcome the defect, consult the changes that need to be made in the .js file of jQuery Mobile, as described in the following Stack Overflow question: IBM Worklight v 5.0.6 - Can't navigate multipages on Windows Phone 7.5 environment (http://stackoverflow.com/questions/17965560/ibm-worklight-v-5-0-6-cant-navigate-multipages-on-windows-phone-7-5-environme)

Loading an external HTML file

An external HTML file is a segment of HTML code that can be injected into any location in the existing DOM. A single HTML file can contain a multiple HTML elements hierarchy.

- JavaScript code or files can be included by using the `script` tag
- CSS files can be included by using the `link` tag
- The HTML file can be injected into the parent element, usually a `DIV`, but this is not mandatory
- jQuery's `$().load()` API method can be used to implement the above

To load an HTML file, use the following syntax:

```
$(containerSelector).load(filePath, callbackFunction);
```

`containerSelector` – jQuery CSS selector of element to host the loaded content. `filePath` – Relative path to an HTML file. Always relative to the main HTML file. `callbackFunction` - a JavaScript function to run when the loading completes.

Example:

JavaScript

```
    $("#pagePort").load(path + "pages/MainPage.html", function(){
    $.getScript(path + "js/MainPage.js", function() {
        if (currentPage.init) {
            currentPage.init();
        }
    });
});
```

HTML

```
<!-- This is a placeholder for dynamic page content -->
<div id="pagePort"></div>
```

This code loads the **MainPage.html** file and inserts its content into the `pagePort DIV` element. JavaScript (and CSS, if needed) references from the **MainPage.html** are loaded into the DOM.

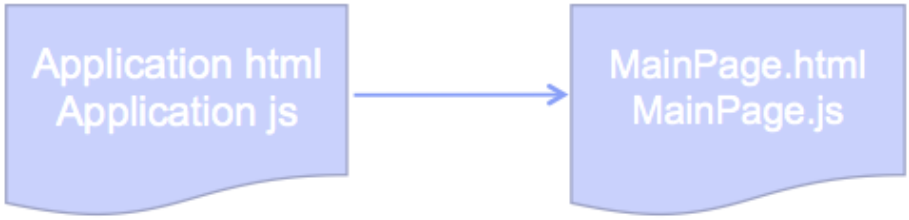
Implementing page navigation with history

By using the previously described technique to load an external HTML file, it is possible to implement a navigation interface with history.

The navigation history must be preserved as a stack in a global array object.
A reference to the currently loaded page must be preserved as well by using a global object variable.
A reference to the file path is needed for Windows Phone 8.

```
var pagesHistory = [];
var currentPage = {};
var path = "";
```

Implementing page navigation with history: Step 1



1. On application launch, **MainPage.html** is loaded from the application code and injected into the `#pagePort DIV`.

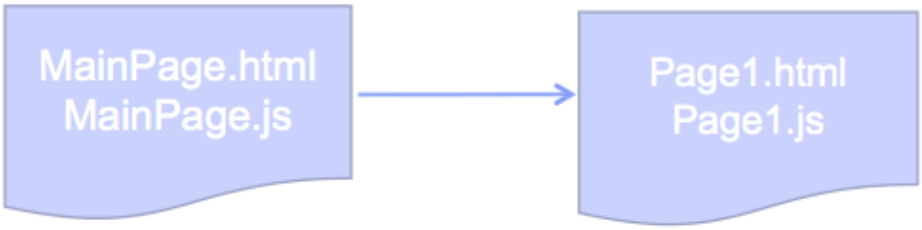
```
function wlCommonInit() {
    // Special case for Windows Phone 8 only.
    if (WL.Client.getEnvironment() == WL.Environment.WINDOWS_PHONE_8)
{
        path = "/www/default/";
    }
    $("#pagePort").load("pages/MainPage.html",function(){
        if (currentPage.init) {
            currentPage.init();
        }
    });
}
```

2. **MainPage.js** is loaded as a part of MainPage.html.

```
currentPage = {};
currentPage.init = function() {
    WL.Logger.debug("MainPage :: init")
;
};
```

3. The currentPage object is declared.
4. The currentPage.init function is declared.
5. When the MainPage.html loading completes, the currentPage.init method is called.

Implementing page navigation with history: Step 2

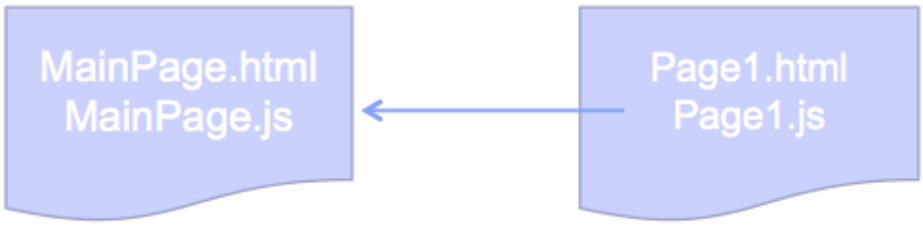


1. **MainPage.html** is pushed into `pagesHistory` stack.
2. **Page1.html** is loaded and injected into `#pagePort DIV` Page1.js is loaded as a part of **Page1.html**.
3. The `currentPage` object is declared and overrides the old one.
4. The `currentPage.init` function is declared.

```
currentPage = {};
currentPage.init = function(){
    WL.Logger.debug(“page1 :: init")
;
};
```

When the loading of **Page1.html** completes, a new `currentPage.init` method is called.

Implementing page navigation with history: Step 3



1. The previous HTML file name is popped from the `PagesHistory` stack (**MainPage.html**).
2. It is loaded and injected into `#pagePort DIV`.

```
currentPage.back = function(){
    WL.Logger.debug("Page1 :: back");
    $("#pagePort").load(pagesHistory.pop()
,
        function() {
          if (currentPage.init) {
            currentPage.init();
          }
    });
};
```

MainPage.js is loaded as a part of **MainPage.html**.

The `currentPage` object is declared and overrides the old one.

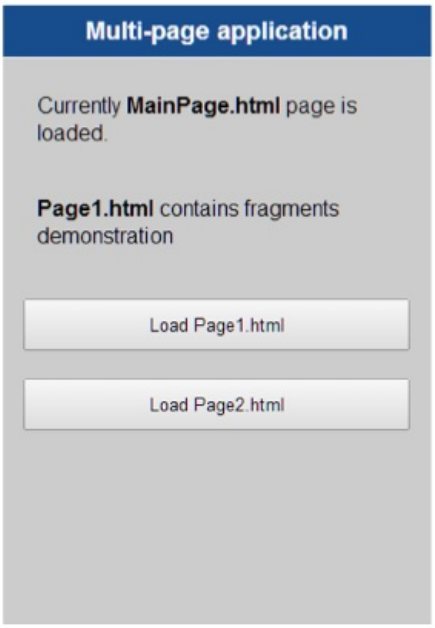
The `currentPage.init` function is declared.

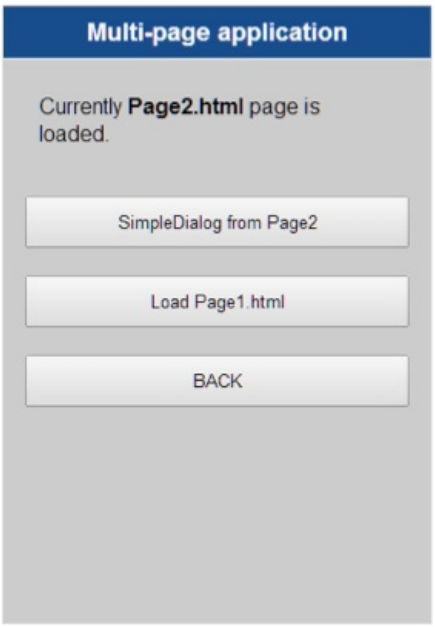
```
currentPage = {};
currentPage.init = function(){
    WL.Logger.debug("MainPage :: init")
;
};
```

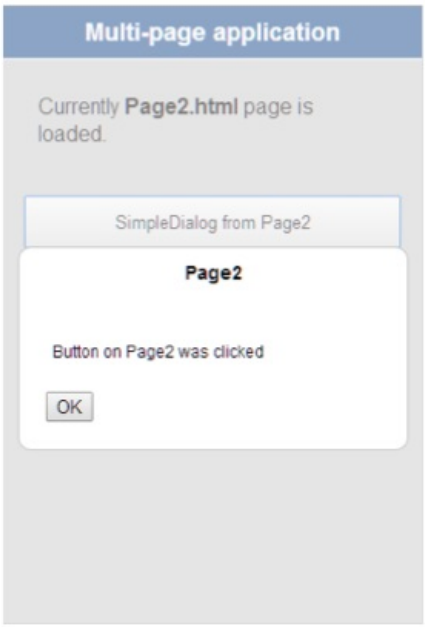
When **MainPage.html** loading completes, the `currentPage.init` method is called.

Sample application

Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/BuildingMultiPageApplicationProject.zip) the Studio project.


Multi-page application
Currently MainPage.html page is loaded.
Page1.html contains fragments demonstration
Load Page1.html
Load Page2.html


Multi-page application
Currently Page2.html page is loaded.
SimpleDialog from Page2
Load Page1.html
BACK


Multi-page application
Currently Page2.html page is loaded.
SimpleDialog from Page2
Page2
Button on Page2 was clicked
OK