Building a multipage application

Overview

This tutorial covers the following topics:

- The basics of a multipage application
- Loading an external HTML file
- Implementing page navigation with history
- Sample application

The basics of a multipage 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 **multipage 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 multipage navigation.

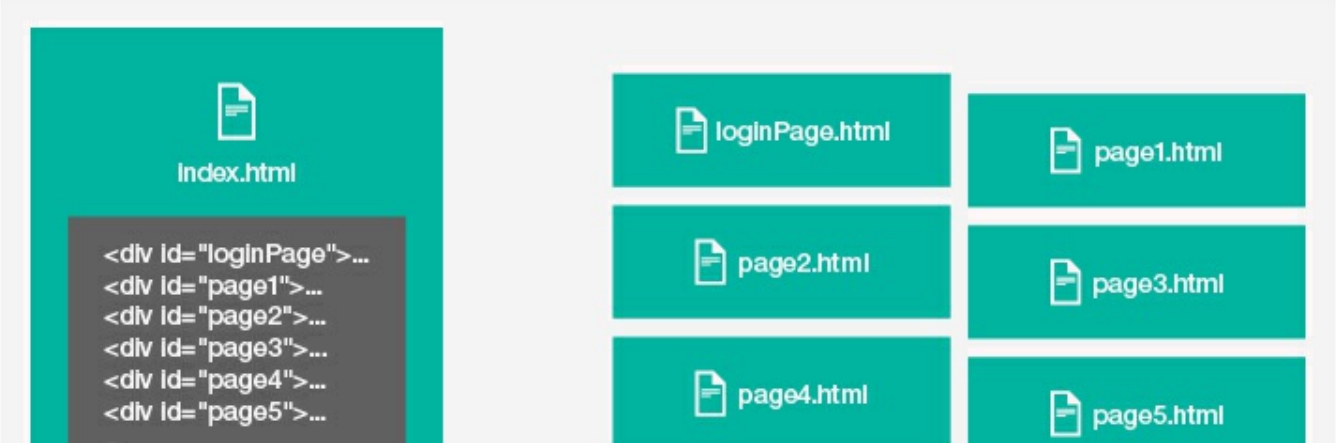
In this tutorial, a multipage MobileFirst application implementation is explained by using **built-in functionality only** with jQuery.

Important: Do not use the built-in functionality that is described in this tutorial if you intend to use a JavaScript UI framework. Use the API of that framework instead.

Building applications with multiple pages

You can build multipage applications in two ways:

- A single HTML file that contains all the application "pages"
- 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 DIVs are hidden at any moment.
- If new content is needed in a page, for example a table, a prepared template cannot be loaded. Instead, 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:

- You can use built-in jQuery APIs to dynamically load, update, and insert DOM elements in your application.
- You can insert HTML pages with CSS and JavaScript as necessary.
- You can implement navigation history.
- You can have JavaScript code executed when pages are loaded or unloaded.

Implementation notes

- When you implement multipage 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 this task in detail.*
  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 you implement 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 hierarchy of multiple HTML elements.

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

To load an HTML file, use the following syntax:

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

`containerSelector` – The jQuery CSS selector of element to host the loaded content.
`filePath` – The 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();
   }
});
});
```

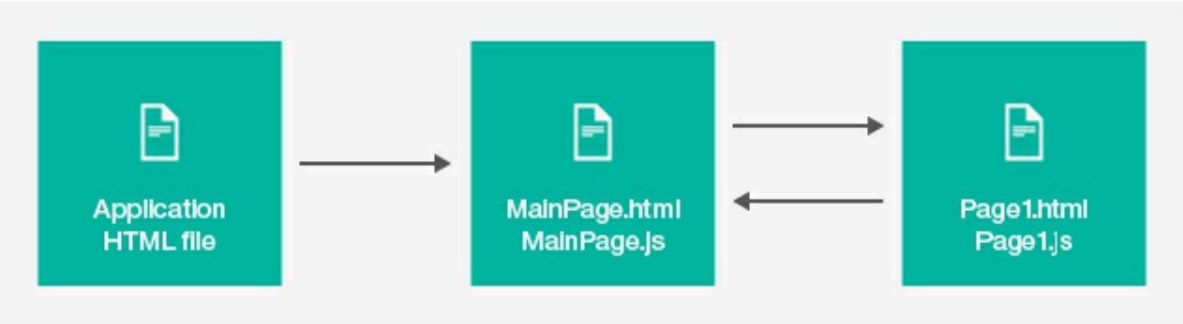
HTML

```
<!-- This is a placeholder for dynamic page content --><br />
<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

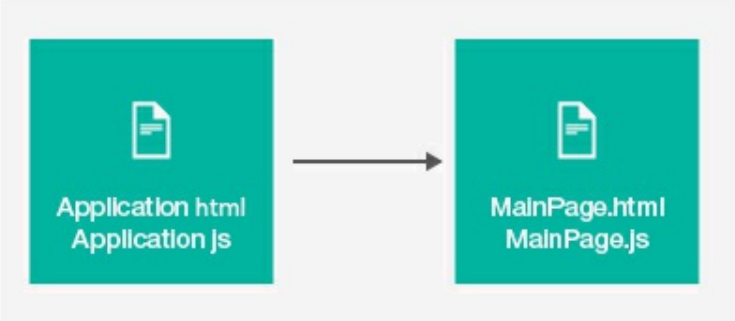
By using the previously described technique to load an external HTML file, you can implement a navigation interface with history.



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

```
 var pagesHistory = [];<br />
var currentPage = {};<br />
var path = "";
```

Implementing page navigation with history: Step 1



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

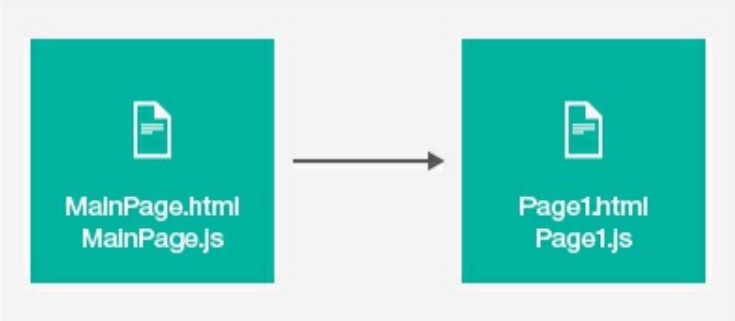
```
function wlCommonInit() {
    // Special case for Windows Phone 8 only.
    if (WL.Client.getEnvironment() == WL.Environment.WINDOWS_PHONE_8)
{
        path = "/www/default/";
    }
    $("#pagePort").load("pages/MainPage.html",function(){
      if (currentPage.init) {
          currentPage.init();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

2. The `MainPage.js` file 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` file is loaded, the `currentPage.init` method is called.

Implementing page navigation with history: Step 2

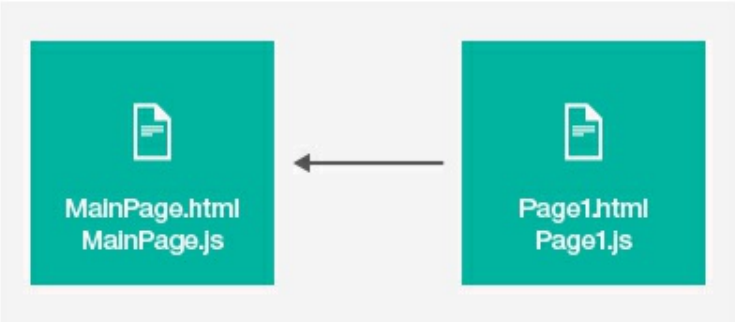


1. `MainPage.html` is pushed into the `pagesHistory` stack.
2. `Page1.html` is loaded and injected into `#pagePort DIV`. `Page1.js` is loaded as a part of `Page1.html`.
3. The `currentPage` object is declared and overrides the old one.
4. The `currentPage.init` function is declared.

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

5. 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 retrieved from the `PagesHistory` stack (`MainPage.html`).
2. It is loaded and injected into `#pagePort DIV`.

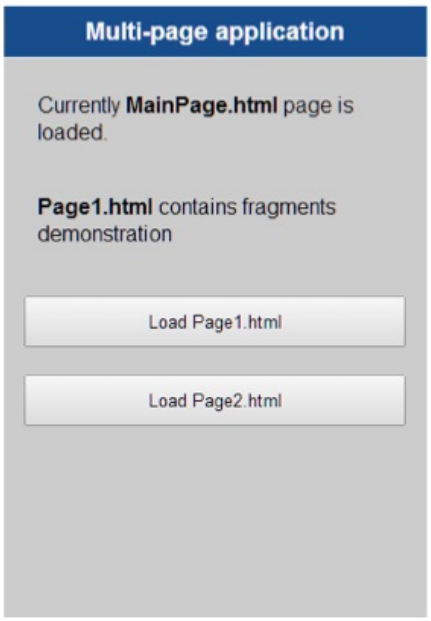
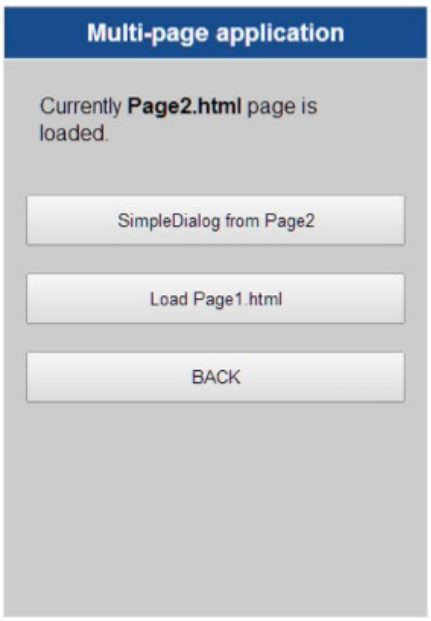
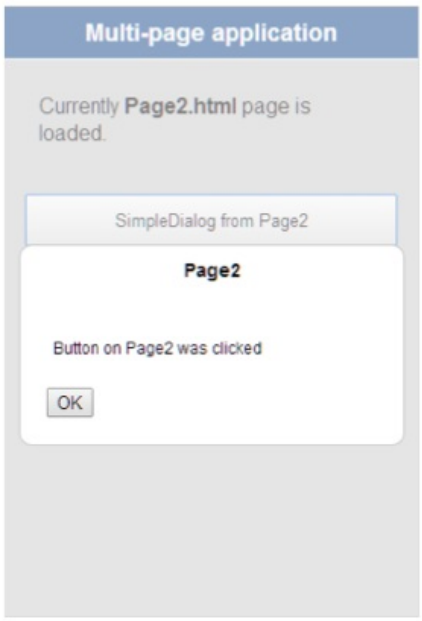
```
currentPage.back = function(){
    WL.Logger.debug("Page1 :: back");
    $("#pagePort").load(pagesHistory.pop()
,
        function() {
            if (currentPage.init) {
                currentPage.init();
            }
    });
};
```

3. `MainPage.js` is loaded as a part of `MainPage.html`.
4. The `currentPage` object is declared and overrides the old one.

5. The `currentPage.init` function is declared.

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

6. When the `MainPage.html` file is loaded, the `currentPage.init` method is called.

Sample application

Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/BuildingMultiPageApplicationProject.zip) the Studio project.