Create a WRT podcast application using the Guarana API

Introduction

01 Aug 2010

In this article you will learn how to create a WRT 1.0 compatible podcast player application that loads XML formatted RSS feed data and MP3 podcasts. You will learn about using Nokia's Guarana API JavaScript framework to create WRT application interfaces, how to use the jQuery framework to conveniently load and parse RSS feeds, and how to manage sound downloads in the WRT environment.

About the Nokia Guarana API

The Guarana API is a set of components based upon the popular jQuery framework and jQuery ui components. It consists of components that are intended for managing user interface issues such as interactive elements for selecting content, displaying content, entering data etc. It also has classes for navigating between screens of information. Using a framework like the Guarana API saves time because you do not have to create your own user interface behaviors and can instead focus on developing an application.

Read the Guarana API wiki post for more information. Guarana UI - a jQuery-Based UI Library for Nokia WRT

Developing a Guarana application

Since the Guarana API is based upon jQuery, you can also use jQuery for other aspects of your application. The example podcast application uses jQuery to load and parse an RSS feed and other tasks.

Download the example podcast application guarana_podcast.zip. here Rename the .zip extension to .wgz to use with a phone or emulator.

Device compatibility

The example code in this article was tested using a Nokia 5800 music Xpress phone with v21.0.025 firmware and with the S60 3rd Edition FP2 emulator. A S60 3rd Edition FP2 phone must have the latest firmware, such as v20.050 for N96 to support the Guarana API.

It is important to note that while the example podcast application is based upon jQuery, there are some jQuery methods which are not supported by the WRT JavaScript interpreter. For backwards compatibility with S60 3rd Edition FP2 devices you should use the `window.onload` event to initiate an application instead of the jQuery `$(document).ready` event and use the standard DOM JavaScript API for appending HTML elements and applying styles instead of jQuery `$.append` and jQuery `$.css` methods.

Application user interface

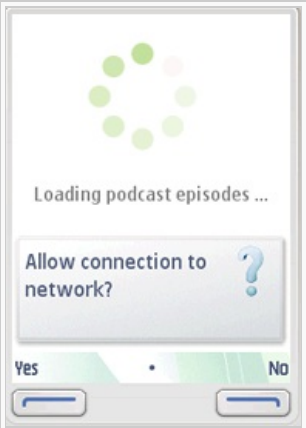
The example podcast application consists of three screens.

A start-up screen to show information about the podcast with a button to initiate download of data listing the current episodes. This screen contains a Guarana Button component to start the process of loading the podcast RSS news feed.

Start-up screen in S60 3rd Edition emulator

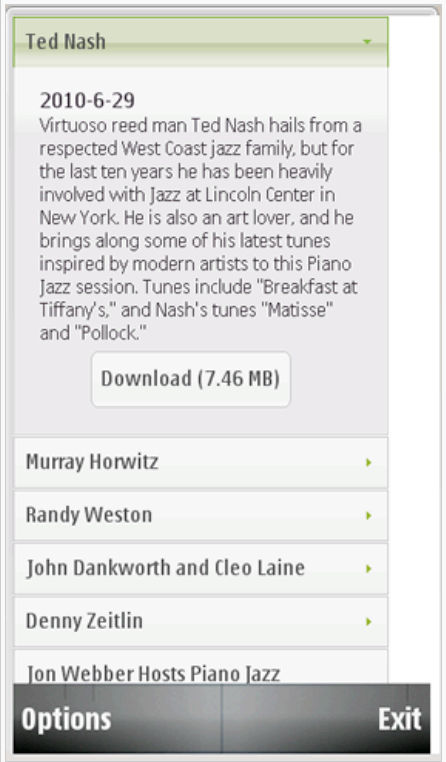
A loading message screen that displays while the data downloads. This screen contains a standard "roller" animation to provide visual interest while the data loads.

(C) Copyright Nokia 2014. All rights reserved.



Loading screen in S60 3rd Edition emulator

An episode screen that displays the list of episodes for the podcast. This screen contains a Guarana accordion menu component to show the list of episodes with expandable and collapsible panes to display an overview of the podcast and a button component within each pane to initiate download of the podcast MP3.



Episode screen in S60 5th Edition emulator

Understanding Guarana Views

Typically a WRT application using the Guarana API will consist of various screens called views. The Guarana API supports a View component and the ViewManager class for conveniently navigating between views. Guarana views are analogous to the deck of cards concept, where each view contains information and user interface components for a given screen and is like a card in a stack. The ViewManager class moves the current view to the top of the deck to show a given screen and hide a previous screen.

Each visible screen in the application has a corresponding Guarana View component. The ViewManager class methods enable the application to display and navigate between views. Views have a corresponding semantic markup in the WRT HTML document. Each view has its own `<div>` tag with a unique id attribute. The `<div>` tag contains the static content and Guarana components for the view.

Before getting into the specifics of creating Guarana views and user interface components you will first learn about the basic HTML structure of a Guarana application.

Guarana HTML template

Download the Guarana API zip archive [Guarana UI: a jQuery-Based UI Library for Nokia WRT] , open the archive and copy the

(C) Copyright Nokia 2014. All rights reserved.

'lib' and 'themes' folders into your WRT project folder with their original directory structure unchanged. The 'lib' folder contains JavaScript files and the 'themes' folder contains the style sheet files necessary to use the Guarana API.

In order to use the Guarana API you must reference several required JavaScript and style sheet files in your HTML file. At this point you may want to review the index.html file from the guarana_podcast.wgz as an example of a Guarana HTML template. The index.html file is located within the guarana_podcast folder in the .wgz archive.

Required CSS

Create your WRT HTML file and add `<link>` tags for the Themeroller.css and custom.css style sheets.

```
<!-- Themeroller CSS -->
<link rel="stylesheet" href="themes/themeroller/default-theme/Themeroller.css"
type="text/css" media="screen">
<!-- Specific Theme/Resolution CSS -->
<link rel="stylesheet" href="themes/nokia/ext-theme/default/360x640/custom.css"
type="text/css" media="screen">
```

The Themeroller.css provides styles for jQuery ui components. These can also be generated from the .jqueryui.com [1] web site to enable different color schemes for your application. The custom.css contains styles specific to the Nokia platform.

Required JavaScript

Add `<script>` tags to load the jquery.js and Guarana.js files. The jquery.js script should be the first JavaScript file loaded into the document. The Guarana.js is dependent upon the jquery.js JavaScript file, so it should load after the jquery.js file.

```
<!-- jQuery file -->
<script src="lib/jquery/jquery.js" type="text/javascript" charset="utf-8"></script>
<!-- Guarana file -->
<script src="lib/Guarana.js" type="text/javascript" charset="utf-8"></script>
```

Tip: You do not need to explicitly load the actual JavaScript and style sheet files for the Guarana API components you use in your WRT application. The Guarana API includes functions to dynamically load the required JavaScript and style sheets for the components you intend to use.

Custom JavaScript and style sheets

In addition to the Guarana and jQuery JavaScript and style sheet files, you may also include the JavaScript and style sheet files for your application. These will most commonly be included after the required Guarana JavaScripts and style sheet files.

For the example podcast application, the custom style sheet file is named “podcastapplication.css” and the custom JavaScript file is called “podcastapplication.js”. Both are located in the guarana_podcast folder of the .wgz archive. You can open the archive to view the HTML, JavaScript, and style sheet code for the application.

Podcast Application HTML for views

Now that you have created the basic HTML template you can begin to define the HTML for the application. In the body section of the HTML document you will first create `<div>` tags with unique id attributes for each view. You can also add static content to these `<div>` tags and include markup for the components contained within each view. The styles for the `<div>` tags are defined in the podcastingapplication.css file located in the root folder of the WRT application archive.

Start-up view HTML

The start up view for the example podcast application consists of an image from the podcast, a brief paragraph describing the podcast and a Guarana Button component which when pressed will execute JavaScript to load the RSS feed and show the most recent episodes for the podcast.

(C) Copyright Nokia 2014. All rights reserved.

Start-up view in S60 3rd Edition emulator

Below is the HTML code for the start up view component, its static content and a Guarana Button component.

```
<div id="startupview">
 <p class="podcastimage"><img src="podcastimage.jpg" /></p>

 <p class="podcastdescription">A preview of upcoming conversations and
   improvisations with Marian McPartland
   and the brightest stars from the world of jazz.</p>

 <div class="btncontainer"><span id="loadrssbtn"></span></div>
</div>
```

The outer `<div>` tag is the element for the view. Note that you should give the `<div>` tag a unique id attribute. Inside the startupview `<div>` tag you can include static content and markup for the button component. The podcastimage.jpg file is included in the guarana_podcast folder of your .wgz archive. The `<span>` tag is the element for the Button component and its outer `<div>` tag is a container to style the button so that it will appear centered.

```
#startupview .btncontainer{
 width:200px;
 margin:auto;
}
```

The `.btncontainer` style sheet class centers the button using the standard technique of setting a width for the element and setting the margin to auto. You can review the remainder of the styles for each view in the podcastapplication.css file.

Loading view HTML

The loading view displays after the user presses the load podcasts button and continues to display while the data loads. This view contains a “roller” gif animation to provide visual interest while the data loads. The view also contains a static text message ending with an ellipsis mark. Below is the HTML code for the loading view component and its static content. Styles for this view center the animation and message. You can use the busyindicator.gif animation stored within the Guarana files or add your animation.

```
<div id="loadingview">
 <p><img src="themes/nokia/base/images/busyindicator.gif" /><p>
 <p>Loading podcast episodes &#8230;</p>
</div>
```

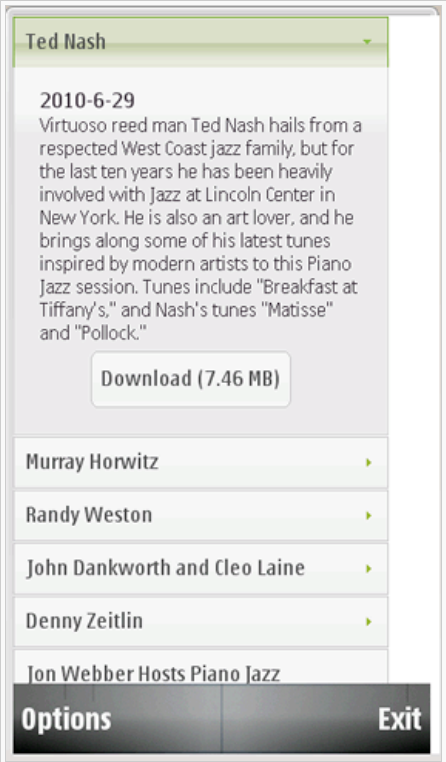
For aesthetic reasons it may be preferable to display the loading message near the top of the screen so that the operating system confirmation messages which appear across the bottom third of the screen, while the phone attempts to connect to the network, will not overlay the animation and message on the loading screen.

(C) Copyright Nokia 2014. All rights reserved.

Loading view in S60 3rd Edition emulator

Episode view HTML

The application displays the the episode view after it completes the process of loading and parsing the RSS feed. The episode view displays the current list of episodes in an accordion menu component. The user can expand or collapse a pane in the menu to read a summary of the podcast. The expanded pane includes a Guarana Button component to start download of the podcast MP3.



Episode view in S60 5th Edition emulator

Below is the HTML code for the episode view component.

```
<div id="episodeview">
 <div id="episodeviewmenu"></div>
</div>
```

Within the episode view `<div>` tag you will include a div tag for the Guarana Accordion component. Since the content of each pane in the accordion menu is dynamically generated during the parsing of the news feed, the application will dynamically generate the HTML markup for each button component within the accordion. Consequently, there is no extra markup required in the HTML document to define the accordion component or the episodeview.

Guarana JavaScript

Next you will learn the JavaScript coding conventions required to dynamically load Guarana components and the order of events

(C) Copyright Nokia 2014. All rights reserved.

required to initialize a Guarana WRT application.

Dynamically loading component JavaScript and style sheets

The Guarana API has the means to dynamically load the JavaScript and style sheet files for the components you intend to use. This streamlines the coding of your application because you do not need to manage the code required to load a specific JavaScript and style sheet for each component.

Guarana global configuration variables

To enable this feature of the Guarana API, you will need to specify the paths to the Guarana JavaScript and style sheet folders within your WRT application using the following Guarana configuration global variables.

```
NOKIA_PATH_JAVASCRIPT sets the path for component JavaScript.
NOKIA_PATH_STYLE_ROOT sets the path for component style sheets.
```

If you copied the Guarana 'lib' and 'themes' folders into your WRT application with no alteration in the directory structure then the paths should be set according to the example below.

```
NOKIA_PATH_JAVASCRIPT = 'lib/';
NOKIA_PATH_STYLE_ROOT = 'themes/nokia/base/';
```

These two variables are required if you intend to use dynamically loaded Guarana JavaScript and style sheets. Once you have set these variables, your application can call the `Nokia.use` function to load the components. Note that the naming of these variables must be in all caps.

Tip:

You must declare the two global variables before calling the Nokia.use function.

Nokia.use function

The `Nokia.use` function dynamically loads the component or list of Guarana components that you specify. To load a component you pass the name of the component, a string in lower case, as an argument to the `Nokia.use` function. You can list an arbitrary number of components, each as an argument to the function. You can also specify the name of a function object or a function literal, as the final argument, to execute once all of the components files are loaded.

In the following example, the application will load the required JavaScript and style sheet files for the Accordion, Button and ViewManager components and after all files are loaded it will execute a function named 'init'. The final argument should be a function object name or function literal, not a string.

```
Nokia.use('accordion','button','viewmanager', init);
```

Review the Nokia.use function Wiki post for more information.

Tip:

You may need to refer to the 'lib' folder to find the valid name of a given component. The component name that you pass to the `Nokia.use` function is the file name of the corresponding JavaScript and style sheet files.

Order of events to initialize the application

Since your application is loading a number of JavaScript and style sheet assets it is important to wait until all files are loaded before initializing the application.

Your application will use the `window.onload` event to wait until its primary assets are loaded, such as the required Guarana JavaScript and style sheets, your application JavaScript and style sheets and any graphics. After these are loaded then your application will execute the Nokia.use function to load files for the specified Guarana components.

(C) Copyright Nokia 2014. All rights reserved.

Once the component files are loaded then the `Nokia.use` function will execute the initialization function you specify to start up the application. The code within the initialization function is usually a script encapsulated in a function that performs various tasks to prepare the application for user interaction once all application assets are loaded and ready.

The following code example demonstrates the order of events to initialize the WRT application using Guarana components.

```
/*
wait until document is loaded and ready before executing any JavaScript
use onload event for compatibility with S60 3rd Edition devices
*/
window.onload = function(){
 Nokia.use('accordion','button','viewmanager', init);
}

// called from Nokia.use
function init(){
…
}
```

Tip:

The jQuery `$(document).ready` event is not backward compatible with S60 3rd Edition FP2 devices. For backward compatibility, Nokia recommends using the `window.onload` event instead.

Podcast Application JavaScript

At this point you should open the podcastapplication.js file found in the guarana_podcast folder of the .wgz archive to review the code for the example podcast application.

Setting Global Variables

There are three global variables declared at the top section of the podcastapplication.js. The first, `PODCAST_RSS` contains the URL for the podcast. The application uses this global variable where ever it requires the URL for the podcast RSS feed.

The other two, `NOKIA_PATH_JAVASCRIPT`, and `NOKIA_PATH_STYLE_ROOT` are the global variables used by Guarana to set the paths to Guarana component JavaScript files and style sheets.

```
/*
global variable to hold URL to podcast RSS feed
*/
PODCAST_RSS = 'http://www.npr.org/rss/podcast.php?id=510056';

/*
configure paths for guarana to dynamically load component JavaScript and style sheets
*/
NOKIA_PATH_JAVASCRIPT = 'lib/';
NOKIA_PATH_STYLE_ROOT = 'themes/nokia/base/';
```

Initializing the application

The next section of code sets up the `window.onload` event to call the `Nokia.use` function to load the files for the Accordion, Button, and ViewManager components. After the components load the code will execute the init function to initialize the application.

```
/*
wait until document is loaded and ready before executing any JavaScript
use onload event for compatibility with S60 3rd Edition devices
```


(C) Copyright Nokia 2014. All rights reserved.

```
 */
 window.onload = function(){
    /*
  dynamically load Guarana JavaScript and style sheet for required components
  wait until files are loaded to initialize application
  */
  Nokia.use('accordion','button','viewmanager', init);
 }
```

init function

The `init` function performs several tasks to get application ready for user interaction. First it calls the `configureWidget` function to configure the WRT environment. Then it calls functions to create views and lastly uses JavaScript to set styles for touch screen devices to enable accordion menu scrolling.

Configuring the WRT environment

The `configureWidget` function calls WRT specific APIs for non touch screen devices so that a device will use key based navigation instead of pointer based navigation, and also to display the soft key labels. Users with a non touch screen device will probably find key based navigation easier to use than pointer navigation.

```
// called from init, configure widget environment and menu
function configureWidget(){
 if(window.widget){
  widget.setNavigationEnabled(false); // for keypress devices, use keypress navigation
  menu.showSoftkeys(); // show the softkey pane
 }
}
```

Tip:

To prevent errors when testing in non WRT browser environments, use an if/then condition to check that the browser environment supports the widget object before executing WRT specific APIs.

Using the Guarana ViewManager

The `init` function starts the process of creating Guarana view components. To manage views, you must first create an instance of the Guarana ViewManager class. The purpose of the ViewManager is to control the display of views.

```
// create Guarana view components
ViewManager = new Nokia.ViewManager(); // global object for the ViewManager
```

You will use the global `ViewManager` instance throughout the application to display and hide views. The next lines of code call functions which create each view for the podcast application.

```
createStartUpView(); // creates the view for the startup screen
createEpisodeView(); // creates the view containing the podcast episodes
createLoadingView(); // creates the view containing the loading message and roller
animation
```

The next line uses the `ViewManager.show` method to display the `startUpView`.

```
ViewManager.show(startUpView); // displays the startUpView
```

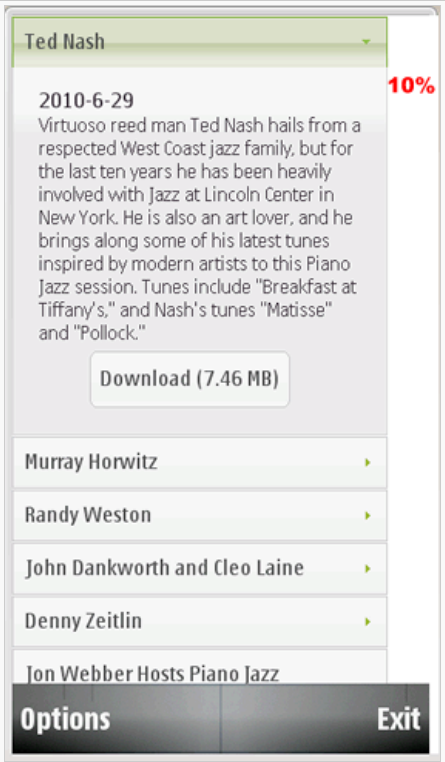

(C) Copyright Nokia 2014. All rights reserved.

This line of code enables the application to automatically display the startup view after the WRT environment has finished loading all assets, so the user will see the startup view when the application first starts.

Review the ViewManager component Nokia Developer Wiki post View Manager for more information.

Enabling menu scrolling on touch screen devices

The final code statement in the `init` function detects touch screen support and dynamically sets a 10% margin on right side of the `episodeviewmenu` element. The `episodeviewmenu` element contains an Accordion menu component. Normally the accordion menu displays in full width across the screen, which does not provide any space for finger dragging to scroll the menu up and down. By setting a 10% margin, the user of a touch screen device has a space to to the right of the menu to touch the screen and scroll the accordion menu up and down without accidentally opening an accordion pane.



Episode view menu with 10% right margin for touch screen scrolling

The following if/then statement changes the style of the `episodeviewmenu` element to support accordion menu scrolling for touch screen devices.

```
// for touch screen devices, create space to the right of menu for finger drag and
scroll
if(typeof device != "undefined"){
 document.getElementById('episodeviewmenu').style.width = '90%';
 document.getElementById('episodeviewmenu').style.marginWidth = '10%';
}
```

Checking for the WRT `device` object is a common technique to detect touch screen support. Devices running the S60 3rd Edition FP2 platform will return false for the `device` object and do not execute the code within this condition and consequently display the accordion menu in full width across the screen. Devices running the S60 5th Edition platform and later implicitly support touch screen, and will execute the code within the condition to display the accordion menu with a finger width margin on the right for up and down scrolling.

Tip:

Use standard DOM API to dynamically set styles because the jQuery `$.css` method is not supported by the S60 5th Edition WRT JavaScript interpreter.

(C) Copyright Nokia 2014. All rights reserved.

Creating Guarana view components

The `init` function calls the `createStartUpView`, `createLoadingView` and `createEpisodeView` functions to create the View components. These functions follow the same three step process to create a Guarana View component. At this point you may want to review the code for the `createEpisodeView` function as an example of how to create a Guarana View component.

Review the View component Nokia Developer Wiki post GuaranaUI View Object for more information.

Extending the view class

The first step in creating a View component is to create a class for a given view by extending the `Nokia.view` class.

It is probably best to think of the `Nokia.view` class as an abstract class that must be extended to form more specific types of classes which you then use to create object instances for your application. For simple applications you should only need to copy the code as it is listed below and just change the class name to something appropriate for your application.

```
var EpisodeView = Nokia.View.extend({
 init: function() {

 },

 show: function() {
  this.getContainer().show();
 }
});
```

The extended class has an `init` event function that executes when the application creates an instance of the view and a show event function which executes when the `ViewManager` class displays the view. In most cases the `show` event function should execute the `this.getContainer().show()` method. Without this line the application will not display the view.

Creating an instance of the extended view class

Next you will create an instance of the extended class. The instance `episodeView` will be the object that you pass to the `ViewManager` to manipulate the view.

```
episodeView = new EpisodeView({
 container: '#episodeview'
});
```

The `container` property must be a valid id selector, which is the id attribute of a `<div>` tag in your HTML document, including the leading # (pound sign). In the above example, 'episodeview' is value of the id attribute of the corresponding `<div>` tag for the view, and the # sign indicates that the name is an id selector for the purposes of jQuery DOM parsing.

Tip:

Make sure that the view component instance is a global variable so you can reference it from other functions if needed.

Adding a view instance to the ViewManager stack

The final step is to add the instance to the `ViewManager` stack. Note that you are adding an instance of the extended class to the `ViewManager`, not the extended class itself.

```
ViewManager.add(episodeView);
```

In the example, the `episodeView` object will the be the object you pass to the `ViewManager` to display or hide this view, or any other manipulation of the episode view.

(C) Copyright Nokia 2014. All rights reserved.

Creating a Button component within a view

The startup view from example podcast application has a Guarana Button component to load the RSS news feed. Unlike the abstract View and ViewManager components, the Button component is a visual component. For most visual components, you will render a pre-existing HTML element into a Guarana component. As you learned previously, the startupview `<div>` tag contains a `<span>` tag, with the unique id “loadrssbtn”. This is element for the Button component.

```
<div id="startupview">
 <p class="podcastimage"><img src="podcastimage.jpg" /></p>

 <p class="podcastdescription">A preview of upcoming conversations and
   improvisations with Marian McPartland
   and the brightest stars from the world of jazz.</p>

 <div class="btncontainer"><span id="loadrssbtn"></span></div>
</div>
```

To add the the Button component to the `startUpView` you will make use of the `StartUpView` class init event function. Inside the `init` function you will add code to create the Button component. When the application creates the view it will execute the init function which then creates the Button component for the “loadrssbtn” `<span>` element.

```
init: function() {
 //Creating Button
 var btn1 = new Nokia.Button({
  element: '#loadrssbtn',
  label: 'List Podcast Episodes',
  click: function() {
   load_rss_feed(PODCAST_RSS); // start loading RSS
   ViewManager.show(loadingView); // display loading message while data loads
  }
 });
}
```

A Button component has a number of configurable properties. For the example podcast application you will set the element, label and click properties.

`element`
: the name of the HTML element to be rendered into the Button component. It must be a jQuery id selector including the leading “#” pound sign.

`label`
: text to appear within the Button component.

`click`
: event function defines what to do when the user clicks the button. In this case the application will call the `load_rss_feed` function to start loading the podcast news feed, and call the `ViewManager` to show the `loadingView` while the RSS feed loads.

Review the Button component Nokia Developer Wiki post [Button] for more information.

Loading RSS

When the user clicks the `loadrssbtn`, the application will call the `load_rss_feed` to load the URL stored in the `PODCAST_RSS` global variable. The `load_rss_feed` function makes use of the jQuery `$.ajax` method to conveniently load the RSS feed.

```
// load the RSS feed, then call parse_rss_feed to parse XML into HTML
function load_rss_feed(feed_url){
 // for Aptana Studio WRT plug-in or testing in web browser
 try { netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRead"); }
```


(C) Copyright Nokia 2014. All rights reserved.

```
catch (e) {};

 // make Ajax call to load RSS feed
 $.ajax({
  url: feed_url,
  success: function(data) {parse_rss_feed(data,'episodeviewmenu')},
  cache:false,
  dataType:'xml'
 });
}
```

The `try catch` statement enables the Mozilla Firefox browser to make AJAX calls to domains different than the one hosting the JavaScript. This is only necessary if you intend to test your application using Firefox or in the Aptana studio environment. Aptana studio uses the Firefox browser to emulate the WRT environment so it is necessary to have this line of code to enable AJAX calls in the Aptana studio testing environment.

Note: Nokia Web Tools have replaced Aptana as the recommended IDE.

The next code block is a standard use of the jQuery `$.ajax` method. The method accepts an object with various properties for loading a file. Once the file loads jQuery will execute the code assigned to the success event, which will call `parse_rss_feed` function.

Review the jQuery documentation[http://jquery.com/] for more information about the `$.ajax` and other jQuery methods mentioned in this article.

Parsing RSS

The `parse_rss_feed` function extracts the data from the podcast news feed, formats the data into HTML and appends this HTML to the `episodeviewmenu` element. It also creates Button components for each item in the news feed and renders the `episodeviewmenu` element into an Accordion component. After it completes these tasks it calls the `ViewManager` to show the `episodeview` so that users can view summaries of each podcast and download the podcast MP3.

Parsing Item tags in RSS

The ultimate goal of the `parse_rss_feed` function is to list each podcast episode as a pane in an Accordion menu component. Each pane in the menu corresponds to an `<item>` tag in the RSS feed and will have a title, date, description and a Button component to download the MP3 for the podcast.

A RSS feed consists of `<item>` tags that contain the data for each episode in a podcast. The `parse_rss_feed` function uses the jQuery `$.find` method to find all of the `<item>` tags in the RSS feed. It then uses the jQuery `$.each` method to loop through each of the `<item>` tags in the RSS feed.

```
$(data).find('item').each(function(item_count){
...
}
```

The `data` variable which was passed into the function from the jQuery `$.ajax` method, represents the XML file for the RRS feed. When the parser reaches an `<item>` tag, it executes the code within the function literal. The `item_count` variable counts the number of times through the loop.

Inside the loop the function extracts data from the current `<item>` tag and stores it properties of an object called `feed_data`. The title, publish date and description for each podcast episode are formatted in `<title>`, `<pubDate>` and `<description>` tags, which are inside the current `<item>` tag.

```
<item>
      <title>Ted Nash</title>
      <description>episode description</description>
```


(C) Copyright Nokia 2014. All rights reserved.

```
      <pubDate>Tue, 29 Jun 2010 17:13:08 -0400</pubDate>
 ...
```

To extract this data, you can use the jQuery `$.find` method to find the corresponding tag within the `<item>` tag and the jQuery `$.text` method to retrieve the data for that tag.

Note that `curitem` is a reference to the current `<item>` tag in the RSS feed.

```
var feed_data = { // create an object to hold data for each rss feed item
 title: curitem.find('title').text(),
 description: curitem.find('description').text(),
 pub_date: curitem.find('pubDate').text()
 …
```

The values for the podcast sound url and file size are stored in the url and length attributes of the `<enclosure>` tag.

```
<enclosure url="http://urlto/some.mp3" length="7825618" type="audio/mpeg"/>
```

You can use a combination of the jQuery `$.find` and `$.attr` methods to retrieve the information for these values.

```
sound_file_url: curitem.find('enclosure').attr('url'),
sound_file_size: curitem.find('enclosure').attr('length'),
```

The `feed_data` object also stores user friendly formatted versions of the date and the file size using the `formatDate` and `getReadableFileSize` functions which are listed at the end of the podcastapplication.js file.

The final property in the `feed_data` object is the `button_id_attr`, which is the unique id for the element that you will render into a Button component.

```
btn_id_attr: 'download_btn' + item_count
```

After extracting the required information for the current podcast episode and storing this information in the `feed_data` object, the `parse_rss_feed` function passes the `feed_data` object to the `appendRSSItem` function.

Appending data to HTML

The next step is to add the information from the current podcast episode to the accordion menu. Each time jQuery moves through the loop and extracts data from the current `<item>` tag, you will build the HTML for one pane and add it to the accordion menu. Each pane will correspond to an episode in the podcast.

To build the HTML for the accordion menu the jQuery loop calls the `appendRSSItem` function. It has two arguments, the first is the `feed_data` object containing the information for the current `<item>` tag and the second is the HTML element name for the accordion menu. The `appendRSSItem` function will format the feed data into HTML and append it to the accordion menu element.

```
// append HTML for this RSS feed item to the accordion container
appendRSSItem(feed_data,accordion_id);
```

Note that the jQuery `$.append` method is not backwards compatible with S60 3rd Edition FP2 devices. For backwards compatibility with s60 v3.2 devices, the `appendRSSItem` function uses standard DOM APIs to build and append HTML.

The `appendRSSItem` function generates the following HTML code for each episode in the RSS feed and appends this to the `episodeviewmenu` HTML element. These blocks of HTML correspond to each pane in the menu.

(C) Copyright Nokia 2014. All rights reserved.

```
<a>title</a>
<!-- description container -->
<div>
 <p class="episodedate">2011-6-28</p>
 <p class="episodedescription">description</p>
 <!-- button container for centering button -->
 <div class="btncontainer">
  <!-- button element -->
  <span id="generated by JavaScript feedObj.btn_id_attr"></span>
 </div>
</div>
```

Each pane in the accordion menu will have a unique title, a description corresponding to each episode. The `<span>` tag is the element for the Button component and it will have a unique id based upon the value of the `feedObj.btn_id_attr`. Note that, like the Button component for the `startupview`, the button element has a container element `<div>` tag with the class `btncontainer` which will apply the styles to center the button within the accordion menu pane.

Creating Button components for the menu

After building and appending HTML for the current episode to the `episodeviewmenu` HTML element, the code execution returns to the `parse_rss_feed` function, which then renders the button HTML element for the current podcast episode into a Guarana Button component.

```
// render the button element into a Guarana API button component
DownloadBtns[item_count] = new Nokia.Button({
 element: '#' + feed_data.btn_id_attr,
 label: 'Download (' + feed_data.sound_file_size_formatted + ')',
 click: function() {
  // prevent errors in non WRT environment
  if (window.widget) {
   widget.openURL(feed_data.sound_file_url);
  }
 }
});
```

Each button component instance is stored in the global `DownloadBtns[]` array. Generally, you should create global variables to represent components in case you need to reference them from other functions. Each Button component has an `element` property which is the unique id attribute with the added '#' sign for jQuery parsing, a label element which lists the file size of the download, and the click event which calls the `widget.openURL` method to load the MP3 for the podcast.

Creating the accordion menu component

After completing the parsing of all items in the RSS feed, appending HTML and creating each Button component, the `parse_rss_feed` now renders the Accordion component.

```
/*
render the accordion element into a Guarana API accordion component
*/
RSSAcordion = new Nokia.Accordion({
 element: accordion_element,
 multiple: false,
 closed: true
});
```

The Accordion component has a variety of configurable properties. For the example podcast application you will assign a jQuery id selector to the `element` property, set the `multiple` property to `false` and the `closed` property to `true`. When set to `false`, the

(C) Copyright Nokia 2014. All rights reserved.

`multiple` property will only have one pane open at a time. When the `closed` property is set to `true` it configures the menu to default to a closed state when the user first uses the menu.

Read the Accordion component wiki post Accordion for more information.

Showing the episodeview

Once the `parse_rss_feed` function has completed rendering the Accordion component, it displays the `episodeview` by calling the `ViewManager.show` method.

```
//show episode view
ViewManager.show(episodeView);
```

This line of code moves the `episodeView` instance to the top of the stack and hides the `loadingView`. Now the user can interact with the accordion menu to open each pane in the menu, read the description of each podcast and start the download of the MP3 for a podcast by clicking on a download button.

Initiating sound download

Current versions of the WRT environment do not support a dedicated audio API for controlling sound embedded in a WRT application screen. So instead of embedding sound, you will offer a better end user experience by passing the URL of the sound to the `widget.openURL` method and allowing the phone music player to download and play the sound.

The music player enables the user to monitor the progress of the file download, control sound playback and save the podcast to the podcast folder in the gallery. Furthermore, the music player for the S60 5th Edition will progressively load the sound, playing the sound as it downloads for a quicker listening experience.

About author

Author, **Hayden Porter**, is a web developer with a special interest in developing for mobile devices. He has written extensively about developing mobile content including white papers for leading mobile device manufacturers and articles for publications such as Electronic Musician Magazine, Music Education Technology Magazine, and DevX.com. For more information, visit www.aviarts.com.

(C) Copyright Nokia 2014. All rights reserved.