F O R U M N O K I A

Nokia s60 WRT 1.0 Podcast Application

Version 1.0; August 4, 2009

WRT 1.0

NOKIA

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

Contents

1 Introduction 6

1.1 Device Compatibility 6

1.2 Before you begin 6

2 Developing a WRT application 6

2.1 Using the Nokia SDK emulators 7

2.1.1 Installing applications with the emulator 7

2.1.2 Testing applications with the emulator 7

2.2 Testing applications on a device 8

2.3 Using the Nokia WRT plug-in for Aptana Studio 8

2.4 Preparing an application icon 8

2.5 About the Nokia WRTKit framework 8

2.6 Extending WRTKit 8

3 Planning the application 10

3.1 User interface 10

4 Initiating the application 12

5 Managing WRTKit controls 13

5.1 Understanding controls on the home screen 13

5.2 Checking for first visit to the home screen 14

5.3 Checking for changes to the favorite podcast list 14

5.4 Creating the “view categories” and “read help” button 15

5.5 Removing the previous favorites menu 15

5.6 Building a new favorites menu 15

6 Navigating through menu screens 16

6.1 Understanding the Listener model 16

6.2 Processing the menu selection 17

7 Accessing data and sound 17

8 Displaying podcast categories 17

8.1 Loading `OPML` data 18

8.2 Displaying a WRTKit notification 18

8.3 Using the prototype.js `AJAX` class 19

8.4 Parsing the `OPML` data 20

8.4.1 Understanding the `OPML` data structure 20

8.4.2 Traversing `OPML` 21

8.4.3 Reading category `outline` tags 21

8.4.4 Reading channel `outline` tags 22

8.5 Building the category menu 22

9 Displaying episodes 23

9.1 Getting the RSS feed URL 24
9.2 Loading the RSS data 24
9.3 Parsing the RSS data 25
9.3.1 Understanding the RSS data structure 25
9.3.2 Traversing RSS 25
9.3.3 Parsing text nodes 26
9.3.4 Parsing tag attributes 26
9.3.5 Formatting data 27
9.4 Building the episodes menu 27
9.4.1 Reusing pre-existing controls 27
9.4.2 Removing extra controls 28
10 Initiating sound download 28
11 Using JSON to store favorite podcasts 29
11.1 Understanding JSON formatting 30
11.2 Saving favorite channels as a JSON formatted string 31
11.3 Converting the JSON formatted string to an object 32
12 Other features of the NPR podcast application 33
12.1 Removing favorites 33
12.2 Setting up modal help 34
12.3 Adjusting header line lengths for orientation 34
12.4 Setting up a modal options menu 34
13 Conclusion 34
About the author 35
Evaluate this resource 36

Change history

Month day, year	Version 1.0	Initial document release

1 Introduction

The Nokia s60 Web Run Time (WRT) platform enables web developers to create standalone mobile applications for Nokia s60 phones. Developers can leverage their existing knowledge of HTML, CSS and JavaScript to build applications that launch from an icon in the Applications folder, run in full screen and integrate with the device through proprietary JavaScript apis. Because of its web "centric" nature, WRT is an idiomatic platform for developing RSS news readers, podcast players, mashups and other types of web applications.

In this article you will learn how to create a WRT 1.0 compatible podcast player application that loads publicly accessible XML data and mobile friendly MP3 podcasts from the National Public Radio network (NPR). You will learn about using Nokia's WRTKit JavaScript framework to create WRT application interfaces, how to use prototype.js to conveniently load XML and interact with JSON formatted data, and how to manage sound downloads in the WRT environment.

1.1 Device Compatibility

Nokia s60 5th edition phones and greater, and Nokia s60 3rd edition fp2 devices with S60 Browser 7.1 support WRT version 1.1. All Nokia s60 3rd edition fp2 phones and greater support WRT version 1.0. The following Nokia s60 3rd edition fp1 phones also support WRT 1.0 with the most recent firmware upgrade: E51, E63, E66, E71, E90, N82, N95, N95-3 NAM and N95 8GB. For a complete list of supporting phones visit the following URLs

http://www.s60.com/life/thisiss60/s60indetail/technologiesandfeatures/webruntime

http://www.forum.nokia.com/devices/matrix_webruntime_1.html

I successfully tested the example NPR podcast application on 5800 Xpress Music, N96, N95 8gb devices and the emulators for the Nokia s60 3rd edition fp2 v1.1 SDK and the Nokia s60 5th edition v1.0 SDK.

1.2 Before you begin

You should have familiarity with JavaScript, HTML and CSS coding and a basic understanding of the Nokia WRT platform.

Review the WRT documentation [http://www.forum.nokia.com/Technology_Topics/Web_Technologies/Web_Runtime/]

Review the WRTtKit framework documentation, available in the Aptana Studio documentation

Download the s60 3rd edition fp2 v1.1 SDK [http://www.forum.nokia.com/info/sw.nokia.com/id/ec866fab-4b76-49f6-b5a5-af0631419e9c/S60_All_in_One_SDKs.html]

Download the s60 5th edition v1.0 SDK [http://www.forum.nokia.com/info/sw.nokia.com/id/ec866fab-4b76-49f6-b5a5-af0631419e9c/S60_All_in_One_SDKs.html]

Download Aptana studio and Nokia WRT plug-in for testing and deployment [http://www.forum.nokia.com/Tools_Docs_and_Code/Tools/Runtimes/Web_Runtime/]

For the sake of brevity this article does not explain all aspects of the NPR podcast application and instead focuses on core features of a basic podcast application including loading data, parsing data, building user interfaces and loading sound. All assets for the application are located in the NPR Podcasts folder included with the download. You should view certain code files such as the application.js and the manage_favorites.js files while reading through sections of the article.

2 Developing a WRT application

Even though a WRT application is developed using familiar web technologies, the process of developing, testing and deploying a WRT application is often different than building a conventional web site.

A WRT application is a JavaScript application. It requires only one HTML file to initiate the application and serve as a "canvas" for DOM scripting to dynamically build user interfaces.

Your application user interface should follow the conventions of a native s60 application consisting of a series of menu screens that "drill down" to detailed information and using the left softkey options menu and the right softkey button to navigate through application screens.

You must package and deploy WRT applications to the emulator or a device for testing. An application package consists of a configuration file, HTML file, icon, JavaScript, CSS, images and media files contained in zip compressed folder with the file extension renamed to .wgz.

You must run your WRT application either in the series 60 SDK emulator or on an actual device to test JavaScript code using the proprietary apis like the widget object.

2.1 Using the Nokia SDK emulators

The Nokia SDK emulators enable you to test WRT applications in a desktop PC environment, which speed up the development process because you do not have to first package and deploy the application to a device for testing.

2.1.1 Installing applications with the emulator

Launch the s60 v3 fp2 emulator or s60 v5 1.0 emulator

Use the open command from the file menu, browse to your .wgz file and select the file.

The emulator will install the application.

2.1.2 Testing applications with the emulator

Open the folder containing your working WRT project files

Open the folder in the emulator directory containing the previously installed WRT application.

C:\S60\devices\S60_3rd_FP2_SDK_v1.1\epoc32\winscw\c\private\10282822

C:\S60\devices\S60_5th_Edition_SDK_v1.0\epoc32\winscw\c\private\10282822

Note: The folder name of the application in the SDK directory has the same name as the Identifier value in the info.plist configuration file for example com.aviarts.npr_podcast.

Copy files from your project folder into the WRT application emulator folder.

Restart the WRT application in the emulator to test changes.

Tip: Save the emulator folder as a windows favorite so you can easily access it later from the favorites menu available from any folder window menu.

Tip: You can enable JavaScript debugging in the s60 5th edition SDK WRT environment by enabling JavaScript debugging in the s60 browser. Start the browser, go to Options->Settings->General and set the Java/ECMA script error item to "Show pop-up notes".

Tip: It is a good idea to test your WRT application in multiple SDK emulators because of differences in the WebKit browser capabilities and operating systems.

2.2 Testing applications on a device

Once you have thoroughly tested the application in the SDK emulator you will need to test the application on the device to ensure that the application works as you expect in the context of mobile device usage. You can transfer over bluetooth, data cable or web browser download.

Tip: Enable `JavaScript` debugging in the device's s60 5th edition web browser.

2.3 Using the Nokia WRT plug-in for Aptana Studio

Another option for testing and deploying WRT applications, is to use the Nokia WRT plug-in for Aptana Studio. Aptana studio enables you to develop and manage your application in a feature rich IDE and debug `JavaScript` in a WRT emulation like environment through the Firefox browser's superior debugging tools. It also automates the process of packaging your application into a .wgz file and can deploy your .wgz file directly to a phone over a bluetooth connection or to the s60 emulator.

[http://www.forum.nokia.com/Tools_Docs_and_Code/Tools/Runtimes/Web_Runtime/]

Note: You should also test your application in the SDK emulator and a device, because the Nokia WRT plug-in for Aptana Studio uses Firefox web browser to run your application, which may interpret code differently than the WRT environment, which is based upon the WebKit web browser.

2.4 Preparing an application icon

Like other s60 application platforms, a WRT application is accessible from an icon in the applications folder. An WRT application icon should be 88 x 88 pixels and saved in png format. For best quality save as either 32 bit png from Adobe Fireworks or 24 bit png from Adobe Photoshop. To see a newly updated icon in the application menu, you will first need to remove any pre-existing install of your application, and then reinstall the application with the new version of the icon. In some cases you may also need to restart the device after removing the old version of the application before install the application with the new icon. If your icon does not alias well against the wallpaper of the phone, you may decide to use a rounded rectangle for the background of the icon. This may improve the perceived quality of the icon because the aliasing occurs at the edges of the rounded rectangle instead of around an irregularly shaped icon.

Figure 1 NPR podcast application icon

2.5 About the Nokia WRTKit framework

I used the Nokia WRTKit framework to build the user interface for the example NPR podcast application. WRTKit has a set of classes 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 WRTKit saves time because you do not have to create your own user interface behaviors and can focus on developing an application. WRTKit is an object-oriented framework consisting of `Views`, which are screens containing content, and `Controls` which are the interactive elements that make up a `View`.

Review the WRTKit documentation which is available in the documentation for the Nokia plug-in for Aptana Studio.

2.6 Extending WRTKit

I extended the WRTKit framework to support features that either did not work as I needed or were not available. My extension enables WRTKit to keep track of the selected control from a menu screen. When the user returns to a given

screen, WRTKit can show the previously selected control and, if the selected control is not in the viewport, scroll down to the control. It also stores each control from a view in an associative `array` that is indexed by the `id` attribute of the control, so that it is easier to reference a control by its name.

Figure 2 Scroll down the Category screen menu and select the Technology option. Selecting the Technology category opens the Technology channels screen.

Figure 3 From the Technology Channels screen, select the Categories right softkey to return to the Categories screen. Upon returning to the Categories screen, the application automatically scrolls down to the Technology option and highlights it.

To support these features I extended the WRTKit `addControl`, `insertControl` and `removeControl` methods by prototyping my own methods onto the `ListView` class. My extended methods do a few extra things before passing arguments onto these core WRTKit methods. The `addNewcontrol` and `insertNewControl` methods both add the control to the `control_list` associative `array` for the view, increment the `array`'s `size` property and pass the control instance onto the corresponding core WRTKit method to be added to the view. The `removeExistingControl` deletes the control instance from the `control_list array`, decrements its `size` property and passes the control instance onto the `removeControl` method to be removed from the view.

To enable automatic scrolling, the `scrollToSelectedControl` method determines if the control is in the viewport. If not, then it scrolls the screen to the selected control by setting calling `document.getElementById(id ).scrollIntoView(true)` so the browser "jumps" to the selected control. This method also sets a style sheet class for the selected control that visually indicates the selected status of the control.

Below is a brief explanation of the extended WRTKit methods that I used in the NPR podcast application code.

`getSelectedControl` – returns the instance of the currently selected control for the view.

`setSelectedControl` – stores the `id` of the currently selected control for the view.

`getControlList` – returns an `object` acting as an associative `array` of controls for the view. The `array` is indexed by the `id` attribute of each control. The `object` has a `size` property indicating the number of controls in the `array`.

`addNewControl` – adds a new control to the view and adds the control to the `control_list` associative `array`. This method has the same arguments as the `addControl` method.

`insertNewControl` – inserts a new control before the specified control in a View. This method has the same arguments as the `insertControl` method.

`removeExistingControl` – Removes the specified control from the View. This method has the same arguments as the `removeControl` method.

`scrollToSelectedControl` – Sets the class attribute of the selected control to `ControlAssemblyFocus` (see WRTKit `CSS` in UI folder) and scrolls to the currently selected control if the control is not in the WRT viewport. This method requires prototype.js functions to check the position of the control in relation to the view port.

Review the code in the WRTKit_extended.js file found within the NPR Podcast folder for more specifics on how to extend the WRTKit.

3 Planning the application

I selected the NPR network for the WRT podcast application example because it has publicly available `XML` news feeds and mobile friendly podcasts. Many of the podcasts are 5mb or less which are reasonable downloads over WiFi and 3G networks.

NPR organizes podcasts by categories, channels and episodes. Categories group channels together by topic. For example, all technology related channels are grouped under the “Technology” category. A channel is analogous to a radio program. Episodes are the shows available for a given radio program channel.

3.1 User interface

Because of the hierarchical organization of information, I developed the user interface for the application as a series of menu screens that enable a user to navigate from the category list at the top of the hierarchy, through the selected channel and its episode list down to the details of a selected episode. I also enabled the user to save a podcast channel as a favorite, which are easily accessible from the home screen.

Figure 4 The categories screen lists all of the NPR podcast categories.

Figure 5 The channels screen displays podcast channels for the selected category.

Figure 6 The episodes screen displays the available podcast episodes for the selected podcast channel.

Figure 7 The episode details screen displays information about a selected podcast episode and a button for downloading the MP3 podcast.

4 Initiating the application

Typically you will code a WRT application to automatically start from a function called by the `window.onload` event in the starting `HTML` file. It is a good idea to wait until the WRT environment loads all of the initially required `JavaScript`, `CSS` and other files before starting the application itself. The example NPR podcast application starts from a function named `init`, defined in application.js, which gets called from the `window.onload` event in the `body` tag of the start.`HTML` document.

```
<body onload="init();">
```

At this point you should review the code for the `init` function located near the beginning of the application.js file.

The `init` function configures the WRT environment and sets instances for various WRTKit classes.

The block of code below configures the WRT `widget` and `menu` objects. It sets navigation to cursor style, displays the softkey pane, specifies an event handler function for the `menu.onShow` event, and calls the `createOptionsMenu` function to populate the items in the options menu.

```
if (window.widget) { // only for widget environment
   widget.setNavigationEnabled(true); // use cursor navigation
   menu.showSoftkeys(); // show the softkey pane
   // handler for dynamically displaying menu items depending upon view
   menu.onShow = menupaneOpen;
   createOptionsMenu(); // build the options menu
}
```

Tip: Using cursor based navigation, by setting `setNavigationEnabled(true)`, ensures forward compatibility with Nokia s60 5th edition touch screen devices.

Note that the `widget object` and the `menu object` apis are proprietary to the WRT environment and are not recognized by desktop PC browsers. Code using these apis tested in a non WRT environment will create error messages. It is a good idea to use `if then` statements to check that these objects exist to prevent errors in a desktop PC browser testing environment.

The following lines of code define instances for the WRTKit framework.

```
// create UI manager
uiManager = new UIManager();
```

The line above defines an instance of the `UIManager` class. WRTKit uses the `UIManager` class to get and set the current view and display status messages.

WRTKit views are screens that display information in the application. You will create a view for each screen in the application. This following code block defines the `ListView` instances for the application and gives each `ListView` a value for its `id` attribute and the view title, also called a `caption`.

```
// create views
// start up screen when no stored favorites
startupView = new ListView("startupView", NPR_LOGO + " Podcasts");
// podcast category view
categoryView  = new ListView("categoryView ", NPR_LOGO + " Podcast
Categories");
//  podcast channel view
channelView = new ListView("channelView", "Podcast Channels");
// podcast episode view
episodeView= new ListView("episodeView", "Episodes");
```

```
// podcast episode view
episodeDetailView = new ListView("episodeDetailView", "Episode Detail");
// help view
helpView = new ListView("helpView", "Help");
// Manage Favorites view
manageFavoritesView = new ListView("manageFavoritesView", "Manage
Favorites");
```

Lastly, the init function calls the `showStartupView` function to display the home screen.

```
// show start up view
showStartupView();
```

5 Managing WRTKit controls

In some cases a WRT application using the WRTKit framework will need to dynamically change the controls in a view. The `showStartUpView` function, which displays the home screen, is an example demonstrating how a WRT application can dynamically build a user interface using WRTKit controls.

At this point you can review the group of functions that manage the home screen from the application.js file starting near line 64.

5.1 Understanding controls on the home screen

The home screen is the first screen that the user sees when the application starts. It has two sets of controls; a menu listing favorite podcasts, and buttons to view podcast categories and read help. The "View Podcast Categories" and "Read Help" buttons are always part of the home screen, and only need to be created once. When the user first uses the application there are no favorites, so the user will only see these two buttons. After the user saves favorite podcasts then the favorite podcast listing will also appear on the home screen. However, the favorites list can change if the user adds or removes a favorite podcast and the application must rebuild the menu, when the user revisits the home page, to show the most recent version of the favorite podcast list.

Figure 8 Home screen displaying View Podcast Categories and Read Help buttons.

Figure 9 Home screen displaying favorites menu in addition to buttons.

The `showStartUpView` function builds the home screen by dynamically adding controls to the `startupView`. This function uses several local and global variables to manage adding controls to, or removing controls from the `startupView`. These variables help determine whether it is the first visit to the home screen, or if the user has changed the favorite podcast list, requiring the application to rebuild the favorite podcast menu.

5.2 Checking for first visit to the home screen

The first line of code in the `showStartUpView` function retrieves all of the controls for the view using the WRTKit extended method `getControlList` and stores these in a variable named `controls`.

```
var controls = startupView.getControlList();
```

The variable `controls` is an `object` referencing each control as an associative `array` indexed by the control `id` attribute.

The variable `first_display` represents whether or not this is the first time the application has displayed the home view. The `boolean` value of `first_display` is true if the global variable `HOME_SCREEN_LAST_VISIT` is `0` and `false` if other than `0`.

```
// true or false for first visit to this view
var first_display = (HOME_SCREEN_LAST_VISIT == 0);
```

The variable `HOME_SCREEN_LAST_VISIT` is set to `0` at the start of the application. The `showStartUpView` function updates the value of `HOME_SCREEN_LAST_VISIT` with a millisecond timestamp at the end of the function code, so it is always greater than `0` after the application first displays the home screen. Therefore, `first_display` should only have a value of `true` the first time the home screen displays and will be `false` for subsequent displays.

5.3 Checking for changes to the favorite podcast list

The next line of code in the `showStartUpView` function sets a `boolean` value for the local variable `hasFavoritesListChanged`.

```
// favorites changed since last visit
var hasFavoritesListChanged = (FAVORITES_LAST_UPDATE >
HOME_SCREEN_LAST_VISIT);
```

The conditional statement compares the millisecond values of the global variables `FAVORITES_LAST_UPDATE` and `HOME_SCREEN_LAST_VISIT`. The application updates `FAVORITES_LAST_UPDATE` with a millisecond timestamp anytime the user adds or removes a podcast from the favorites list. If the timestamp of

FAVORITES_LAST_UPDATE is greater than the last visit to the home screen then the user has changed the favorites list since the last visit to the home screen. The showStartUpView function will use the hasFavoritesListChanged variable to determine whether or not to rebuild the favorite podcast menu.

5.4 Creating the "view categories" and "read help" button

The following block of code from the showStartUpView function creates the "View categories" and "Read Help" buttons, only when the application first displays the home screen. Otherwise, these buttons already exist and there is no need to re-execute the code. Note that re-executing this code would unnecessarily duplicate these two buttons.

```
// create buttons for view podcast categories and read help
// only done once, on first display of the view
if (first_display == true) { // no controls yet added to the view
   // create view podcast categories
   var catbtn = new FormButton("viewcategories", "View Podcast
Categories");
   catbtn.addEventListener("ActionPerformed", showOrLoadCategories);
   startupView.addNewControl(catbtn);
```

WRTKit FormButton controls have an HTML id attribute, a caption, which is the label for the button and an event handler function. Generally, you will use FormButton controls for executing specific actions in a WRT application.

To create the control, the showStartUpView function first creates a new instance of the FormButton Class, give it the id "viewcategories", and the caption "View Podcast Categories'. The showStartUpView function defines the event handling function using the addEventListener method. When the user selects the "view podcast categories" button, JavaScript will execute the function showOrLoadCategories to display the category list in a new screen. Lastly, showStartUpView function adds the control to the startupView with the WRTKit extended addNewControl method. Note, that there is similar code for creating the "Read help" button.

5.5 Removing the previous favorites menu

Next, the showStartUpView function checks if the favorite podcasts list has changed and if so, removes any pre-existing favorite podcast buttons to prevent duplication of controls when rebuilding the menu.

```
// remove pre-existing favorites buttons if favorites list changed
if (hasFavoritesListChanged == true) {
   // some favorites buttons already exist
   for (id in controls) {
      // skip these buttons
      if (id != "viewcategories" && id != "viewhelp") {
         // remove control
         startupView.removeExistingControl(controls[id ]);
      }
   }
}
```

The code checks that hasFavoritesListChanged is true, meaning that the favorite podcasts list has changed since the last visit to the home screen. If the user has changed the list, the code loops through each control and removes all controls from startupView except the view categories and read help buttons.

5.6 Building a new favorites menu

The showStartUpView function is now ready to create the favorite podcast list menu. For efficiency, the function should only build the menu on the first display of the home screen, or if the user has changed the favorite podcast list.

```
// only build menu for first display of home screen
```

```
// or if favorites list has changed
if(first_display == true || hasFavoritesListChanged == true){
```

The code then checks if there are any favorites using the `getFavoritesPref` function which returns `null` if there are no stored podcasts or a `JSON` formatted `string` representing the stored favorite podcasts. You will learn more about the process of saving favorite podcasts in section 11.

```
   // if there are favorites then build menu
   // attempt to get list of favorites
   var favorites_json = getFavoritesPref();
   if (favorites_json != null) { // pref contains saved favorites
```

The `getFavoritesAsObject` function converts the `JSON string` into an `array` of `JavaScript` objects. The `showStartUpView` function then builds the menu with the most recently added favorite podcast at the top of the menu. Each option in the menu is a `NavigationButton` type of control.

```
      // get favorites as a JavaScript object
      var podcast_favorites = getFavoritesAsObject();
      // build menu listing favorites, display favorites so that most
recent are first
      var index_start = podcast_favorites.length - 1;
      for (var i = index_start; i >= 0; i--) {
         // get the channel title
         var btn_caption = podcast_favorites[i].title;
         // create the control
         var favorite_btn = new NavigationButton(i, RSS_ICON,
btn_caption);
         // add an event
         favorite_btn.addEventListener("ActionPerformed",
channelSelected );
         // insert the control
         startupView.insertNewControl(favorite_btn,
controls['viewcategories']);
      }
```

Each `NavigationButton` control has a numeric value for its `id` attribute, an `RSS` icon graphic and a `caption` as taken from the title of the corresponding favorite podcast. The global variable `RSS_ICON` contains the path to the `RSS` feed icon graphic stored inside the .wgz archive.

The `showStartUpView` function also assigns the `channelSelected` function as the event function handler for each favorite podcast button. When the user selects a favorite podcast from the menu, the application will execute the `channelSelected` function to display the current episodes for the podcast. The code uses `insertNewControl` to add the control before the view categories button. This builds the favorite podcast menu above the view categories and read help buttons so the menu is easily accessible at the top of the home screen.

6 Navigating through menu screens

Before delving into other aspects of the application, it is helpful to understand how the application manages navigation through the various menu screens. To access podcasts, a user browses from the categories screen down to the episode details screen. Each screen has a menu enabling the user to “drill down” to more specific information. When the user selects an option in the menu, `JavaScript` dynamically builds the assets for the next screen so the user can access the next level of information. To do this, the application needs to know which control the user selected and how to process the selection event.

6.1 Understanding the Listener model

An application using the WRTKit framework is based upon an event listener model. When a user selects a control, `JavaScript` executes the event handler function assigned to the control’s event listener callback method. The

controls for the categoryView, channelView and episodeView screens all have specific event handler functions for processing events from a menu selection.

For example, each NavigationButton control in the category menu on the categoryView screen has the categorySelected function as the event handler function assigned to its event listener.

```
categorybtn.addEventListener("ActionPerformed", categorySelected);
```

When the user selects a category from the menu, JavaScript executes the categorySelected function.

6.2 Processing the menu selection

The categorySelected function receives an event object from a button's event listener callback function. The event.source object represents the menu option control that the user selected.

```
// button event handler
function categorySelected(event){
   categoryView.setSelectedControl(event.source.id );
   showChannelView();
}
```

Each NavigationButton control has a unique numeric value for its id attribute. The categorySelected function gets the id of the selected control from the event.sourc.id property and stores the value of the id attribute using the WRTKit extended setSelectedControl method. The application will retrieve this numeric id later to access the channels for the selected category from the global PODCAST_CATEGORIES object, and also display the selected menu item when the user returns to the categoryView screen. The function then calls the showChannelView function which builds the menu for the channelView screen.

7 Accessing data and sound

NPR provides a publicly available OPML formatted category list that includes all of the channels grouped by category. OPML is a XML markup intended for outlining topics and sub topics and is a common format for creating podcast category listings.

Each channel has a URL to its corresponding RSS feed that lists current episodes. RSS is a standard XML markup for listing posts to a blog or podcast. Each item in the RSS feed includes information about the episode and a URL to download the MP3 sound file for the episode.

8 Displaying podcast categories

To display podcast categories, the application must download the OPML data and dynamically build the category menu on the categories screen from the data. Since the category list does not change very often, it is only necessary to download the OPML data and create the category menu once during the application session. All subsequent visits to the categoryView screen will not require reloading the OPML data and rebuilding the menu.

At this point you can review the group of functions that manage the categoryView screen from the application.js file starting near line 134.

The showOrLoadCategories function determines whether to load the OPML data and build the category menu, or just show the already existing category menu.

```
// check to see if the categoryView  has any content
function showOrLoadCategories(){
   var controls = categoryView .getControlList();
   if(controls.size > 0){
      showCategoryView(); // has existing content, show category view
```

```
    } else {
       loadPodCastCategories(); // no content yet, load categories
    }
}
```

In the first line of code, the function gets the list of controls for the `categoryView`. If the `size` property is greater than `0` then the category menu already exists because the view already has controls assigned to it. In this case the function calls the `showCategoryView` function to display the view without reloading the `OPML` data and rebuilding the menu.

If the `size` property equals `0` then there are no controls and no category menu. The function calls the `loadPodCastCategories` function to start the process of loading the `OPML` data, parsing the data into a `JavaScript object` and building the category menu user interface.

8.1 Loading `OPML` data

The `loadPodCastCategories` handles loading the `OPML` data. The function first determines the `URL` of the `OPML` file based upon whether the application is configured to load the online file or the local file. The `URLs` to these two files are defined in the global variables section at the beginning of the application.js file.

```
if(TEST_OFFLINE == true){
   var opmlurl = OPML_LOCAL; // load local copy
} else {
   // always load a new copy of online opml
   var opmlurl = addNoCacheStr(OPML_ONLINE);
}
```

When loading the online `OPML` file, the application calls the `addNoCacheStr` function to append a timestamp as a name value pair at the end of the `URL`. Below is an example of how the `addNoCacheStr` function modifies the `URL` to the `OPML` file.

```
http://www.npr.org/podcasts.opml?nocache=1234567
```

The millisecond timestamp ensures that the `URL` is always unique requiring the WRT environment to load the most recent online version of the file instead of a previously cached version of the `OPML` file.

8.2 Displaying a WRTKit notification

The `loadPodCastCategories` function then displays a notification, using the WRTKit `uiManager.showNotification` method to alert the user that the application is attempting to load the category list over the Internet.

Figure 10 Show a notification while the categories OPML file downloads.

```
uiManager.showNotification(-1,"wait","loading categories",-1); // show
notification
```

Typically you will use the "wait" type of WRTKit notification for alerts that last for an undetermined length of time. This line of code configures an alert with the message "loading categories" that will remain visible for an indefinite period of time. The application will hide the alert after the OPML file finishes loading or if there is a failure to load the OPML file.

Note: Review the WRTKit documentation in the Aptana Studio help files for more information on using WRTKit notification messages.

8.3 Using the prototype.js AJAX class

Next, the loadPodCastCategories function makes an AJAX call to load the OPML data from the NPR web site using the prototype.js Ajax class. The prototype.js Ajax.Request method requires a URL to the OPML document, and an object that sets the HTTP method of the request to "get", and assigns function literals to the onSuccess and onFailure callback functions.

```
new Ajax.Request(opmlurl,
{
method:'get',
      onSuccess: function(transport){
         var xmlDoc = transport.responseXML;
         uiManager.hideNotification();
         populateCategoryView(xmlDoc);
      },

      onFailure: function(){
         uiManager.showNotification(3000,"warning","Unable to load
categories.",0);
      }
   }
);
```

After the file loads, the prototype.js Ajax.Request class calls the function assigned to the onSuccess call back.

```
onSuccess: function(transport){
   var xmlDoc = transport.responseXML;
   uiManager.hideNotification();
   populateCategoryView(xmlDoc);
```

```
}
```

This code passes the XML object representation of the OPML document to the populateCategoryView function to parse the document and build the category menu. At this point the application exits the loadPodCastCategories function and executes the populateCategoryView function.

If the OPML fails to load, then the Ajax.Request class calls the onFailure call back function which uses the WRTKit uiManager.showNotification method to display a message "Unable to load categories", alerting the user to the problem. In this case the application will not proceed to the populateCategoryView function.

```
onFailure: function(){
   uiManager.showNotification(3000,"warning","Unable to load
categories.",0);
}
```

You can use the "warning" type of WRTKit notification to alert users to errors in your application. The value of 3000 milliseconds sets the duration of the notification to 3 seconds.

For each call to the showNotification method, the WRTKit uiManager will replace any previously displayed notification with the new one, so there is no need to explicitly hide the "loading categories" message before showing the "unable to load categories" message.

Note: Review the online prototypejs documentation for more information on the Ajax class
http://prototypejs.org/api/ajax/request

8.4 Parsing the OPML data

After loading the data, the application must now parse the data into a form that is usable for JavaScript. The populateCategoryView passes the XML object to the getPodCastCategories function which handles parsing of the OPML data.

8.4.1 Understanding the OPML data structure

NPR organizes podcast channels within its OPML file both by radio station and by topic. To simplify the application user interface I decided to limit the menu system to browsing by topic only. Consequently, the parser code will need to skip all of the radio station outline tags and only display the data from category outline tags. The parser also needs to get information for the channel outline tags nested within each category outline tag.

Below is an excerpt of the OPML showing first the outline tag "BySource" which lists podcasts by radio station, followed by individual outline tags for each topic category.

```
<body>
<outline text="BySource">
<outline text="Colorado Public Radio">
<outline text="CPR: Colorado Matters" title="CPR: Colorado Matters"
type="rss" version="RSS" xmlUrl="http://www.npr.org/rss/podcast.php?id
=510072" htmlUrl="http://cpr.org/co_matters/"/>
      </outline>
   …
   (more outline tags for each local NPR station or network)

   (the following outline tags group podcast channels by category)
<outline text="Arts &amp; Entertainment">
<outline text="NPR: In Character" title="NPR: In Character" type="rss"
version="RSS" xmlUrl="http://www.npr.org/rss/podcast.php?id =17914370"
htmlUrl="http://www.npr.org/blogs/incharacter/"/>
…  (more channels for the Arts and Entertainment category)
```

```
   </outline>
   … ( more categories)
</body>
```

Each category `outline` tag has a `text` attribute representing the name of the category such as “Arts & Entertainment” and nested channel `outline` tags containing information about the radio show channel for a category.

```
<outline text="Arts &amp; Entertainment">
```

Each channel `outline` tag has a number of attributes including the `title` of the radio show and the `URL` to the `RSS` feed for the show.

```
<outline text="NPR: In Character" title="NPR: In Character"
type="rss" version="RSS"
xmlUrl="http://www.npr.org/rss/podcast.php?id =17914370"
htmlUrl="http://www.npr.org/blogs/incharacter/"/>
```

8.4.2 Traversing `OPML`

The `getPodCastCategories` function traverses the `outline` tags of the `OPML` document, gets the required attribute values of each tag and returns a two-dimensional `array` of objects representing categories and their respective channels. Each category `object` has a `text` property for the name of the category and a `channels array` which is collection of objects represented the channels for a given category. Each `channel object` has a `title` property for the name of the radio show and an `xmlurl` property for the `URL` to the radio show `RSS` feed.

The first few lines of code in the `getPodCastCategories` function define variables.

```
var categories = new Array();
var root = xmlobj.getElementsByTagName('body')[0];
var outlines = root.getElementsByTagName('outline');
```

The `categories array` will contain an `object` for each category. The variable `root` represents the `body` tag, which is the `node` were the function begins traversing the `DOM` of the `OPML` document. The variable `outlines` is an `array` representing all `outline` tags in the `OPML` document.

```
var num_outlines = outlines.length;
var cat_count = 0;
```

The variable `num_outlines` stores the number of outlines and the variable `cat_count` represents the number of categories that have been parsed, starting at `0`.

The `getPodCastCategories` function then uses a nested loop structure to iterate through all of the category `outline` tags and nested channel `outline` tags. The outer loop traverses the category `outline` tags and the inner loop traverses the channel `outline` tags nested inside of each category `outline` tag. Note that the application must skip all of the `outline` tags for radio stations and only parse the category `outline` tags and corresponding channel `outline` tags.

8.4.3 Reading category `outline` tags

The outer loop of the `getPodCastCategories` function handles parsing of the category `outline` tags.

```
for(var i=0; i<num_outlines; i++){
   if(outlines[i].parentNode.tagName == "body" && i>0){
```

All of the category `outline` tags have the body tag as their `parentNode`. The radio station `outline` tags have the “`BySource`” tag as their `parentNode`. The `if then` condition in the loop limits parsing to only the category `outline` tags by only allowing `outline` tags which have the body tag as the `parentNode`. This

eliminates the radio station `outline` tags because these tags have the "`BySource`" `outline` tag as their `parentNode`. Checking that `i>0` skips the "`BySource`" tag itself, which is the first `outline` tag in the document.

Next, the `getPodCastCategories` function begins to build the two dimensional `array` representing the hierarchical relationship of categories and their channels.

```
var outline_text = outlines[i].getAttribute("text");
categories[cat_count] = new Object;
categories[cat_count].text = outline_text;
```

The variable `outline_text` contains the name of the current category. The `DOM getAttribute` method returns the value of the `text` attribute of the current `outline` tag. The `getPodCastCategories` function then creates a new `object` in the category `array` using `cat_count` as the index of the `array`. Each `category object` has its `text` property set to the name of the category. The function increments the value of `cat_count` after each iteration of the outer loop.

Since each category has channels associated with it, the `getPodCastCategories` function creates a second dimension `array` named `channels` to act as a collection of objects representing each channel in a given category.

```
categories[cat_count].channels = new Array();
var podcast_channels = outlines[i].getElementsByTagName('outline');
var num_channels = podcast_channels.length; // store array
```

The variable `podcast_channels` is an `array` representing all channel `outline` tags nested within the current category `outline` tag. The variable `num_channels` stores the number of channels for the current category.

8.4.4 Reading channel `outline` tags

The inner loop of the `getPodCastCategories` function handles parsing of data for the channel `outline` tags.

```
for(var j=0; j<num_channels; j++){
   categories[cat_count].channels[j] = new Object;
   categories[cat_count].channels[j].title =
podcast_channels[j].getAttribute("title");
   categories[cat_count].channels[j].xmlurl =
podcast_channels[j].getAttribute("xmlUrl")
}
```

The code creates a new `Object` for each channel in the category and assigns the `title` and `xmlUrl` attributes from the current channel `outline` tag to the `title` and `xmlurl` properties of the `channel object` for the current channel.

Lastly, the `getPodCastCategories` returns the two-dimensional `array` to the `populateCategoryView` function.

8.5 Building the category menu

The `populateCategoryView` function builds the menu for the `categoryView`. First the function assigns the two-dimensional `array` returned from the `getPodCastCategories` function to the global variable `PODCAST_CATEGORIES`. Note that since the `PODCAST_CATEGORIES` also contains the channel listings and the application will use this same `object` to create the menu for the `channelView`.

```
// object listing categories and channels for each category
PODCAST_CATEGORIES = getPodCastCategories(xmlobject);
```

Next the function loops through the `object`, gets the title of each category and builds `NavigationButtons` for each category.

```
for(var i=0; i<PODCAST_CATEGORIES .length; i++){
   var text = PODCAST_CATEGORIES [i].text;
   var categorybtn = new NavigationButton(i,CATEGORY_ICON,text);
   categoryView .addNewControl(categorybtn);
   categorybtn.addEventListener("ActionPerformed", categorySelected);
}
```

When the user selects a category, the application will call the `categorySelected` function which starts the process of building the channels menu for the channels screen. You can review the functions for building the `channelsView` screen in the application.js file starting near line 260.

Note that the `NavigationButtons` on the category screen have the icon graphic defined in the `CATEGORY_ICON` global variable. I used different icons for the `NavigationButtons` on the category, channel and episodes menu screens so that the end user can visually understand their location in the hierarchy of application interface based upon the icon type that appears on screen.

Figure 11 Icons for categories, channels and podcasts

9 Displaying episodes

To review, the NPR podcast user interface is a series of menu based screens enabling a user to navigate through the hierarchy from categories to channels to episodes to the details of a selected episode. When the user selects a radio show channel on the `channelView` screen, the application takes the user to the `episodeView` screen to see the available episodes for that channel.

The `episodeView` consists of a menu listing the current episodes for the channel and a header with the name of the selected radio show channel.

Figure 12 This episode screen lists all available episodes for the NPR: Technology podcast.

9.1 Getting the RSS feed URL

When a user selects a channel on the channelView screen, JavaScript executes the channelSelected function. The primary purpose of the channelSelected function is to retrieve the RSS feed for the selected channel and pass it to the loadEpisodes function which starts the process of building the episode menu for the episodeView.

It is important to note that the controls for the favorite podcast channels menu on the home screen also call the channelSelected function. So the channelSelected function needs to determine if the user is navigating from the home screen or the categoryView screen, so it can use the appropriate object to get the RSS feed for the channel.

If the user selects a channel from the favorite podcast menu on the home screen then the channelSelected function needs to use the information stored in the WRT preference. Otherwise the function uses the PODCAST_CATEGORIES object to retrieve the RSS URL.

```
// coming from startupView favorites btn
if(uiManager.getView() == startupView){
…
   var selectedFavorite = startupView.getSelectedControl().id ;
   var podcast_favorites = getFavoritesAsObject();
   var episodeUrl = podcast_favorites[selectedFavorite].xmlUrl;
```

The if then statement uses the uiManager.getView method to determine the current view. If the current view is the startUpView then the user is navigating from the home screen to the channelView screen, and the channelSelected function must get the RSS feed from the WRT preference.

The channelSelected function uses the id of the selected control as the index for the podcast_favorites array. The podcast_favorites is an array of objects representing the channels stored in the WRT preference. The structure of this object is the same as the channels object. The function stores the RSS feed URL for this channel in the episodeUrl variable and passes this value to the loadEpisodesfunction.

If the user navigates from the categoryView screen to the channelView screen then the function uses the PODCAST_CATEGORIES object to retrieve the RSS feed.

```
} else { // coming from some other view, not using favorites
   var selectedCategory = categoryView .getSelectedControl().id ;
   var selectedChannel = channelView.getSelectedControl().id ;
   var episodeUrl = PODCAST_CATEGORIES
[selectedCategory].channels[selectedChannel].xmlurl;
}
```

Using a similar process, the channelSelected function uses the id of the selected control as the numeric index to look up the channel object in the array. It then stores the value of xmlurl property containing the RSS feed URL in the episodeUrl and passes this to the loadEpisodesfunction.

9.2 Loading the RSS data

At this point you can review the group of functions that manage the episodelView screen from the application.js file starting near line 325.

Like the loadPodCastCategories function in section 8.3, the loadEpisodes function makes an AJAX call using the prototype.js Ajax class to load the RSS feed XML file for the selected channel. The loadEpisodes function displays a notification that the RSS feed is loading, and the onSuccess callback function passes the XML object representation of the RSS feed to the showEpisodeView function.

9.3 Parsing the RSS data

After loading the data, the application must now parse the data into a form that is usable for JavaScript. The showEpisodeView function passes the XML object representing the RSS feed to the getPodCastEpisodes function to handle parsing of the RSS data. After the getPodCastEpisodes function parses the data it returns an array representing the episodes for the selected channel to the showEpisodeView function which builds the episodes menu.

9.3.1 Understanding the RSS data structure

A RSS feed XML file consists of item tags that contain information about each episode for the radio show. Each item tag has child tags for specific information about the show such as the title, description, publication date and the URL to the MP3 sound file of the podcast. Below is an example item tag from RSS feed for the NPR: In Character Podcast radio show for the episode "Robert Jordan, Hemingway's Bipartisan Hero".

```
<item>
      <title>Robert Jordan, Hemingway's Bipartisan Hero</title>
      <description><![CDATA[Though fierce political opponents, John
McCain and Barack Obama agree on a literary matter: Each picks Ernest
Hemingway's 1940 novel <em>For Whom the Bell Tolls</em>, featuring the
stoic freedom-fighter Robert Jordan, as a favorite.]]></description>
      <pubDate>Tue, 14 Oct 2008 08:03:12 -0400</pubDate>

<link>http://www.npr.org/templates/story/story.php?storyId=95604448&amp;
ft=2&amp;f=17914370</link>
      <guid>http://podcastdownload.npr.org/anon.npr-
podcasts/podcast/17914370/95686256/npr_95686256.MP3</guid>
      <itunes:summary><![CDATA[Though fierce political opponents, John
McCain and Barack Obama agree on a literary matter: Each picks Ernest
Hemingway's 1940 novel For Whom the Bell Tolls, featuring the stoic
freedom-fighter Robert Jordan, as a favorite.]]></itunes:summary>
      <itunes:keywords>NPR,National Public Radio,In Character,Morning
Edition,All Things Considered,Fresh Air</itunes:keywords>
      <itunes:duration>7:19</itunes:duration>
      <itunes:explicit>no</itunes:explicit>
      <enclosure URL="http://podcastdownload.npr.org/anon.npr-
podcasts/podcast/17914370/95686256/npr_95686256.MP3" length="3547932"
type="audio/mpeg"/>
    </item>
```

9.3.2 Traversing RSS

The NPR podcast application, displays information from the title, description, pubdate and enclosure tags. The getPodCastEpisodes function traverses the item tags of the RSS XML document, gets the values of these tags, stores these values in an array of objects and returns the array. Each object in the array has corresponding properties for the title, description, published date, MP3 URL and sound file size of a podcast episode.

The first few lines of code in the getPodCastCategories function define variables for positioning the parser in the XML document.

```
// get a reference to the root-element "rss"
var root = rssfeed.getElementsByTagName('rss')[0];
// get reference to "channel" element
var channels = root.getElementsByTagName("channel");
// get all "item" tags in the channel
var items = channels[0].getElementsByTagName("item");
```

The variable root represents the rss tag, which is the root node of the DOM. The variable channels represents the node of the channels tag and items is an array representing each item tag within the channels tag. These three lines of code position the parser to return the item tags for the first channel tag within the first rss tag. Note that usually there is only one channel in a RSS news feed for a podcast, so the array only has one channel tag.

```
var news_items = new Array(); // array of objects
```

The array news_items will contain an object for each episode.

Next the getPodCastEpisodes function loops through each item in the RSS XML document and creates an object for each item tag to be stored in the news_items array.

```
// set number of items to display
num_news_items = items.length;
for(var i=0; i<num_news_items; i++){
   news_items[i] = new Object;
```

9.3.3 Parsing text nodes

Some of the information for the episode is stored as the text node of a tag, such as the title tag. The text node is the information within the start and end tags. For example in the title tag below the value of the text node is "Robert Jordan, Hemingway's Bipartisan Hero".

```
<title>Robert Jordan, Hemingway's Bipartisan Hero</title>
```

Generally, when parsing a RSS feed you use the statement getElementsByTagName("title")[0] to get the node of the title tag for a given item. More specifically, to retrieve the title of an episode, the getPodCastEpisodes function finds the tags named "title" for the current item using the DOM method getElementsByTagName("title") which returns an array of all of the title tags for this item. To access the first, and only title tag in this item, you reference the array index 0. Typically there is only one title tag for each item in a RSS feed, so the array only has one title.

```
// extract information from rss feed store in array
// as properties of array
news_items[i].title =
items[i].getElementsByTagName("title")[0].firstChild.nodeValue;
```

To read the text node of the title tag, the function accesses the firstChild of the title tag, which is the first item contained within the tag. In this case the firstChild is the text node itself. The getPodCastEpisodes function reads the actual value of the text node from the DOM nodeValue property.

9.3.4 Parsing tag attributes

The MP3 URL and file size are stored in the url and length attributes of the enclosure tag which requires a different syntax to parse.

```
// get url of the sound
news_items[i].soundurl =
items[i].getElementsByTagName("enclosure")[0].getAttribute("url");
}
```

To read the value of an attribute, the getPodCastEpisodes function accesses the enclosure tag using the previously described syntax and retrieves the value of the specified attribute with the DOM getAttribute("url") syntax.

9.3.5 Formatting data

Because the values for the publish date and the file size are not in a simple format, the `getPodCastEpisodes` function passes these two values to other functions to change the information into a more familiar reformat. The `formatDate` function takes the value from the `pubdate` tag and formats it into `yyyy/mm/dd` format.

The file size of the `MP3` is stored as a value in bytes in the `length` property of the `enclosure` tag. The `getPodCastEpisodes` function passes this value to the `getReadableFileSize` function to convert the byte value to kb, mb or gb values which are more meaningful to end users.

9.4 Building the episodes menu

The last step is for the `getPodCastEpisodes` function to call the `showEpisodeView` function to build the episodes menu for the `episodesView` screen. The application should always build a new menu when the user navigates from the `channelView` or a favorite podcast selection on the `startupView`, because a new channel selection requires a new episode menu. However, the application does not need to rebuild the menu if the user returns to the `episodeView` after viewing the details on for an episode because the user may want to view a different episode for the current channel. In this case the `showEpisodeView` function only shows the `episodeView`by calling the `uiManager.setView` method.

The `showEpisodeView` function will proceed to build the menu only if the current view is either the `channelView` or the `startupView`. The following `if then` condition manages this comparison.

```
var current_view = uiManager.getView();
if(current_view == channelView || current_view == startupView){
```

If the user is navigating from one of these two views then the function begins the process of building the menu. The function sets control variables for the loop that builds the menu.

```
PODCAST_EPISODES = getPodCastEpisodes(xmlobject);
   var num_episodes = PODCAST_EPISODES.length;
   var controls = episodeView.getControlList();
```

The global variable `PODCAST_EPISODES` contains the `array` of item objects returned from the `getPodCastEpisodes` function. The application uses a global variable to store this information because it will use the data in this `object` again to build the `episodedetailView` screen. The `num_episodes` variable contains the number of episodes to control the number of times to iterate through the loop and the `controls` variable contains a list of all the controls for the `episodeView`.

9.4.1 Reusing pre-existing controls

Next the `showEpisodeView` function starts building the episodes menu. This loop proceeds through each episode in the `PODCAST_EPISODES` object. For efficiency, the function will reuse existing controls by renaming the `caption` of the control, add any new controls if needed and lastly remove any extra un-needed controls.

```
for(var i=0; i<num_episodes; i++){
   var btn_text = PODCAST_EPISODES[i].title;

   // check to see if button created
   if(controls[i] != null){ // button object exists
      controls[i].setText(btn_text); // change text
```

The function checks if there is a control for a given index in the controls list and if a control already exists, then simply re-label the control using the WRTKit `setText` method to assign the episode to the `caption` of the button.

```
   } else { // cteate button object
```

```
            var episodebtn = new
NavigationButton(i,PODCAST_ICON,btn_text);
            episodeView.addNewControl(episodebtn);
            episodebtn.addEventListener("ActionPerformed",
episodeSelected);
   }
}
```

If there is no pre-existing control at a given index, then the function creates a new button. Each control in the episode menu has a podcasting icon graphic to help the user visually discern that this content pertains to podcast sound files.

9.4.2 Removing extra controls

In some cases there may be more controls than needed for the current menu. For example, a user may have previously browsed to a channel with ten episodes then selected a new channel with only five episodes. In this case the `for loop` described in section 9.4.1 will reuse and re-label the first five controls. However, there are still five controls left over from the previously built menu. Consequently the `showEpisodeView` function will need to remove these extra un-needed controls. The following block of code manages this task.

```
/* check if there are more buttons than needed to display episodes for
this channel */
if(controls.size > num_episodes){
   // store before loop, controls.size decrements
   // with each removeExistingControl call
   var controls_to_remove = controls.size;
   for(var i=i; i<controls_to_remove; i++){
      // remove extra buttons from previous category
      episodeView.removeExistingControl(controls[i]);
   }
}
```

The `if then` statement checks for extra controls by determining if there are more controls in the view than there are episodes to display. Next the function stores the number of controls for the `channelView` in a separate variable `controls_to_remove`, because as it removes controls the `control.size` property decreases and would adversely affect the logic of the loop control. The `for loop` in this code block starts counting where the last `for loop` finished, which is be the numeric `id` of the last needed control. The `showEpisodeView` function calls the WRTKit extended `removeExistingControl` method to remove the control from the menu.

10 Initiating sound download

Current versions of the WRT environment do not support a sophisticated user interface for controlling sound embedded in a WRT application screen. So instead of embedding sound, you will offer a better end user experience by downloading the sound with the `widget.openURL` method and allowing phone music player to launch to 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 Series 60 5th edition devices will progressively load the sound, playing the sound as it downloads for a quicker listening experience.

The NPR podcast application launches the music player when the user selects the “Download Podcast” button on the `episodeDetailView` screen. The event listener for this button calls the `downloadPodcast` function to handle the download. At this point you should review the code for the `downloadPodcast` function near line 499 in the application.js file.

```
function downloadPodcast(){
   // show loading message
   var selectedEpisode = episodeView.getSelectedControl().id ;
```

```
    var soundfileurl = PODCAST_EPISODES[selectedEpisode].soundurl;
    widget.openURL(soundfileurl);
}
```

To determine the `URL` of the podcast, the `downloadPodcast` function first gets the `id` property of the control that the user selected from the `episodeView`. Next the function references the corresponding episode from the `PODCAST_EPISODES array` using the `id` as the index, and storing the `soundurl` property of this episode in the `soundfileurl` variable. Lastly, the function passes the value `soundfileurl` to the `widget.openURL` method which causes the phone to launch the music player and start the podcast download.

11 Using `JSON` to store favorite podcasts

Users have the ability to save the current channel as a favorite podcast by selecting the "Save as Favorite" option from left softkey options menu from the `episodesView` and the `episodeDetailView` screens. The NPR podcast application displays the saved favorite podcasts in the favorites list on the home screen. This is a convenient feature that enables users to quickly access the podcasts that they listen to frequently.

Figure 13 Save a favorite podcast channel from the left softkey options menu.

Figure 14 View the list of favorite podcast channels from the home screen.

The WRT environment saves persistent data in a WRT preference variable, which is a useful way to store user generated data, such as the podcast favorites list, in between sessions of the application. However, current versions of the WRT environment only support storing string data types, not structured native JavaScript data types like objects and arrays.

The favorite podcasts is a list of channels each with a title and a RSS feed URL. This structured data is most efficiently represented as an array of JavaScript objects each with a corresponding title and xmlUrl property. To preserve this structure while saving in a WRT preference, the NPR podcast application stores the favorites list as a JSON formatted string, and uses the included JSON functions of prototype.js to conveniently convert the data between JSON and a JavaScript array.

11.1 Understanding JSON formatting

JSON (JavaScript Object Notation) is a representation of a JavaScript object such as an array or an object type in Object notation format as a string. Most often we see JavaScript objects represented with dot syntax.

Objectname.propertyname = “value”;

However JavaScript also supports object notation to represent an object.

{“propertyname” : ” value”}

By placing this syntax inside a string a developer can quickly convert the string into an object using the JavaScript eval function. The main advantage of using JSON is that it is easy to represent data in a structured human readable way as a string and also quickly convert it to a native JavaScript object instead of parsing the string which is required with XML.

Note: For more information about JSON visit www.json.org.

The NPR podcast application represents each favorite podcast channel as an object in an array of objects. Each object has a title and an xmlUrl property.

```
podcast_favorites[0] = {title:title,xmlUrl:xmlUrl};
```

To save the array in the WRT preference variable, the NPR application converts the array into a JSON formatted string. Below is an example of the JSON string formatting for a podcast favorites list. This example consists of two podcast channel objects each with a title and a xmlUrl property.

```
'[' +
   '{“title”:"NPR: In Character",
   “xmlUrl”:"http://www.npr.org/rss/podcast.php?id =17914370"},' +
   '{“title”:"An Evening With...",
   “xmlUrl”:”http://www.npr.org/rss/podcast.php?id =510189”}
']';
```

The following outline explains the formatting purpose of each character in the JSON string.

```
[ <- left bracket starts the array
    ‘{ <- single quote indicates first element in array, curly brace indicates object type
       “title” <- object property named title
       : <-  colon assigns value to the property
       “NPR: In Character” <- quotes indicate string value of the title property
       , <- comma delimits properties
       “xmlUrl” <- name of next property, xmlUrl
       : <-  colon assigns value to the property
       “http://www.npr.org/rss/podcast.php?id =17914370” <- string value xmlUrl property
    }’ <- curly brace indicates end of object, single quote is end of first array element
    , <- comma delimits items in array
```

```
    '{<- single quote indicates 2nd element in array, curly brace indicates object type
        "title" <- object property named title
        : <-  colon assigns value to the property
        "An Evening With..." <- quotes indicate string value of the title property
        , <- comma delimits properties
        "xmlUrl" <- name of next property, xmlUrl
        : <-  colon assigns value to the property
        "http://www.npr.org/rss/podcast.php?id =510189" <- string value xmlUrl property
    }' <- curly brace indicates end of object, single quote is end of 2nd array element
] <- right bracket is end of array
```

11.2 Saving favorite channels as a JSON formatted string

When a user saves a podcast channel as a favorite podcast, the NPR podcast application first retrieves the current favorites list as a JSON formatted string from a WRT preference, and converts the JSON string into a JavaScript array. Then the application adds the user selected channel to the array, converts the array back into a JSON formatted string and saves the string back into the WRT preference variable. At this point you should review the code for the savePodcastAsFavorite function from the manage_favorites.js file near line 73.

The first block of code in the savePodcastAsFavorite function determines the currently selected podcast channel and retrieves the title and RSS feed URL from the corresponding PODCAST_CATEGORIES object.

```
var selectedCategory = categoryView .getSelectedControl().id ;
var selectedChannel = channelView.getSelectedControl().id ;
```

The selectedCategory and the selectedChannel variables represent the currently selected podcast category and channel.

```
// get values to save from the currently selected channel
var selected_channel = PODCAST_CATEGORIES
[selectedCategory].channels[selectedChannel];
var title = selected_channel.title; // get the selected episode title
var xmlUrl = selected_channel.xmlurl; // get the selected episode URL
```

The savePodcastAsFavorite function uses these two values to look up the object for the channel in the PODCAST_CATEGORIES object and store the corresponding title and RSS feed URL in variables.

Next the savePodcastAsFavorite function calls the getFavoritesAsObject function which retrieves the JSON string for the currently saved favorites from a WRT preference and converts it to a JavaScript object, see section 11.3.

```
// using prototype.js json functions
var podcast_favorites = getFavoritesAsObject(); // get favorites as a
JavaScript object
```

The getFavoritesAsObject returns an array of objects representing the favorite podcast channels and assigns this to the podcast_favorites array.

In the next block of code the savePodcastAsFavorite function calls the isPodcastSaved function to determine whether the current channel is already saved. The isPodcastSaved function compares the title of the channel the user wants to save to those already saved in the favorites list and if there is a match then the function returns true. If the channel is already saved then the code alerts the user using a WRTKit notification, then exits the savePodcastAsFavorite function to avoid duplicating an already saved channel in the favorites list on the home screen.

```
// check that the podcast is not already saved
if(isPodcastSaved(title,podcast_favorites) == true){
    // show message that this podcast is already saved.
```

```
   uiManager.showNotification(1000,"info","You have already saved this
as a favorite.",-1);
   return; // exit function to avoid saving a copy of this podcast
}
```

If the channel is not yet saved then the function adds the channel as an `object` to the `podcast_favorites array`.

```
// otherwise not yet saved, save new favorite title and URL into the
// JavaScript object
if(podcast_favorites.length == 0){ // no previously stored favorites
   // store this favorite as the first in the array
   podcast_favorites[0] = {title:title,xmlUrl:xmlUrl};
} else { // previously stored favorites
   // append this favorite channel to the array
   podcast_favorites.push({title:title,xmlUrl:xmlUrl});
}
```

If there are no favorites in the favorites list, then set the current channel as the first in the favorites list. Otherwise the function uses the `Array.push` method to append the channel to the favorites list. Note that the channel is an `object` with a `title` and `xmlUrl` properties.

Next the `savePodcastAsFavorite` function uses the prototype.js `Object.toJSON` method to convert the updated `podcast_favorites array` to a `JSON` formatted `string` and stores the `string` in the `favorites_json` variable.

```
// resave json string in WRT pref
// using prototype.js json functions
// convert object to a string
var favorites_json = Object.toJSON(podcast_favorites);
// resave the json string into the WRT pref
saveFavoritesPref(favorites_json);
```

The `saveFavoritesPref` function the saves the `JSON string` in a WRT `WRT preference` named "`favorites_json`" and saves the time stamp in the `FAVORITES_LAST_UPDATE` global variable for use in building the favorites list on the home screen.

11.3 Converting the `JSON` formatted `string` to an `object`

When the user views the home screen, the NPR podcast application retrieves the `JSON string` from the `WRT preference` named "`favorite_json`" and uses the prototype.js `evalJSON` method to convert the `string` into an `array` to build the favorites menu list. At this point you should review the `getFavoritesAsObject` and `getFavoritesPref` functions from the manage_favorites.js near the beginning of the code.

```
// convert json formatted string into JavaScript object
// using prototype.js function
function getFavoritesAsObject(){
   // convert the json string in WRT pref into a JavaScript object
   // get the json string from the WRT pref
   var favorites_json = getFavoritesPref();
```

The `getFavoritesAsObject` function calls `getFavoritesPref` function to get the `JSON` formatted string stored in the `WRT preference`. The `getFavoritesAsObject` function stores the value from the `WRT preference` in the variable named `favorites_json`.

```
   if(favorites_json != null){ // favorites already exist
      // using prototype.js json functions
```

```
        // convert json string into JavaScript object, return object
        return favorites_json.evalJSON(true);
```

The function then calls the prototype.js evalJSON method to convert the string in favorites_json to an array and returns this array. The `showStartupView` function described starting in section 5.2 uses the array passed from the `getFavoritesAsObject` function to build the favorites menu for the home screen.

Note: When the optional argument for `evalJSON` is set to true, the `string` is checked for possible malicious attempts and `eval` is not called if one is detected.

If the value of `favorites_json` is `null` then the `WRT preference` is empty and there are no channels yet saved as favorites. In this case the `getFavoritesAsObject` function returns an empty `array`.

```
    } else {
        // return new array if no podcast favorites saved
        return new Array();
    }
}
```

The `getFavoritesPref` function returns the `JSON` formatted `string` stored in the `WRT preference` named "`favorite_json`". If this `WRT preference` is empty or `undefined` the function returns `null`.

```
// attempt to get json formatted string from pref
function getFavoritesPref(){
    // attempt to get list of subscribed feeds
    if(window.widget){ // WRT environment
        // returns undefined if  WRT preference is not yet set
        var pref = widget.preferenceForKey("favorites_json");
        // set pref to null for undefined or ""
        if(pref == "" || typeof pref == "undefined"){pref = null};
        return pref;
    }
}
```

12 Other features of the NPR podcast application

The focus of this article is on loading data and sound, saving podcast channels and exploring WRTKit user interface management. However, there are a few other features of the application which may be of interest. In this section I briefly describe other aspects of the application.

12.1 Removing favorites

In addition to the functions for saving podcast favorites there is also a feature for removing favorites from the list. You can review how the manage favorites feature works by looking at the code in the application.js starting near line 517.

`showManageFavoritesView` – This function displays the `manageFavoritesView` and initiates building the interface to remove favorites.

`buildFavSelectionMenu` – This function builds a menu for selecting the favorites for removal. The menu consists of checkbox items using the WRTKit `SelectionList` class, so a user can select one or more favorites for removal.

`deleteSelectedFavorites` – This function executes when the user selects the "Delete Selected Favorites" button on the manage favorites screen. It converts the existing favorites list from a `JSON` string into an `array`, removes the selected channels from the `array` using the `Array.splice` method, converts the `array back` into a `JSON` string and resaves the `JSON string` in the `WRT preference`.

12.2 Setting up modal help

When a user selects help from the left softkey options menu or on the home screen, the application will show help and instructions that are relevant to the current screen. You can review the functions for this feature in the application.js file starting near line 592.

The `showHelpView` function determines the current view using the `uiManager.getView` method and uses a `switch case` statement to display the appropriate help content for the currently selected screen. It also dynamically assigns a function to the right softkey so that the user can return to the previous screen by selecting "Close Help".

12.3 Adjusting header line lengths for orientation

Some of the podcast channel titles can be too long for display as the header of the episode and episode details screen on a device with a 240 x 320 resolution in portrait mode. This causes an undesirable line break in the header for these screens. I set up code in the application to truncate long captions to prevent line breaks in portrait mode and restore the full caption in landscape mode. You can review the code for this feature starting near line 678 of the application.js file.

To do this, I first extended the WRTKit `ListView` object to have a `fullcaption` property to store the original title of the channel. When the user changes the phone's orientation the browser executes the `window.onresize` event which calls the `resetCaptionLength` function. If the current view is the episode or episode detail view then the `resetCaptionLength` function passes the `fullcaption` of the current view to the `getShorterCaption` function to truncate the line length of the header. The `getShorterCaption` function sets the length of the caption for the episode and episode details screen based upon the number of allowed characters configured in the `CAPTION_LENGTH_240` and `CAPTION_LENGTH_320` global variables.

12.4 Setting up a modal options menu

To make the application easier to use I set up code that dynamically changes the left softkey options menu to display only those options that are relevant to the current screen and application context. You can review the code that configures the options menu from the options_menu.js file found in the NPR podcast folder.

The `init` function configures the menu by assigning the `menupaneOpen` function to the `onShow` event of the menu and calling the `createOptionsMenu` to populate the menu with all of the possible options. When the user selects the left softkey the WRT environment executes the `onShow` event which calls the `menupaneOpen` function. The `menupaneOpen` function determines the current view using the `uiManager.getView` method and dynamically shows the relevant options for this view and hides the others. You can use the setDimmed method of the menu object to show or hide a menu item in the left softkey options menu. Passing a value of `true` hides the menu item, while a value of `false` shows the menu item.

```
menu.getMenuItemById(1).setDimmed(true); // hide this menu item
```

13 Conclusion

The NPR podcast application is an example of WRT platform that enables users to quickly access web based content over the air, download and save this content to the phone gallery and save user generated data in between application sessions. While WRT has a more limited feature set than platforms such as Java or C++, it offers enough capability to support a wide variety of applications, and is especially idiomatic for web oriented content.

The Nokia WRT 1.0 platform should be of interest to web developers who want to use their existing skill set to create mobile applications that integrates with the phone capabilities more than current browser based web applications. Content owners wanting a mobile presence should also consider this technology because it offers users a branded application with quick access to content by launching from an icon, and a customized experience by running in full screen with deeper phone integration compared to a mobile web site.

About the author

Hayden Porter, the author of this document, has been developing web sites professionally since 1995 and has a special interest in developing multimedia content for mobile devices. He has written extensively on the subject of 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, see www.aviarts.com.

Evaluate this resource

Please spare a moment to help us improve documentation quality and recognize the resources you find most valuable, by rating this resource.

*** This chapter is always the last chapter of the document. If the document has an appendix, remove the numbering from the heading of this chapter, as done here. Before publication, complete the survey link by adding in the ID part of the URI generated by the metadata editor. ***