# AppSpot

Framework Specification

# Introduction

Spotify and other music services is excellent. AppSpot is a project which aims to provide a new way to browse, discover and share music. AppSpot will go beyond limitations that exists in the current software regarding online music, when it comes to integration with third party services. It relys on existing RAD-framework technologies such PHP, Ajax, XUL and .NET (for windows platform).

This work is a result from the earlier project that where mentioned here, ’SpotiApps’. It was a experimental project aimed to accomplish ’extra’ views in the Spotify window. However, this create a numberous of issues and was ridiculus complex, so it didn’t go beyond the experimental stadge.

AppSpot will do it. AppSpot will provide a ’Open Source’ alternate to the Spotify client when it comes to browsing. It will combine the three well-known RAD platforms C#, PHP and Mozilla to create a ultimate front-end for music browsing and sharing, in the beginning phase to extend Spotify abilities. It will also share the same UI features as a open source completion of the existing software.

## The framework

AppSpot is a standalone C# application that takes advantage of the GeckoFX library for rendering of custom user interface. As with Spotify, whole application work are around the ’URI’ system. For this time, it uses the ’spotify:’ uri system as a extender, ~~but may be changed in a future release (~~It will be able to use this it in the further with any uri).

The core apporoach of the system is the ’views’ that can be associated with the application. A view is no more than a folder with a XUL file that contains preprocessing PHP code and XUL declarations. Those declarations can then generate search results, playlists and other features that has a shared UI model. Probably the UI model will be able to be customized in a later phase.

### Views

In the AppSpot application directory, a mandatory folder called ’View’ relives. This folder can contains a unlimited number of directories, whose name titles the view that are specified in the file with the ’view’ extension. Those views are preprocessed by a built-in PHP parser that are loaded in the program folders. The reason for using a preprocessor of each XUL view is following:

1. Overcoming the problems with the XMLHttpRequest that lives in the GeckoFX engine. (Workaround see later later)
2. The AppSpot system consists of a URL, where the contents after the third (3) colon resembles a query that are handled by the PHP preprocessing. For now, each URI is consists of one ”scope” (which you can set to whatever, to match links from various service such spotify: etc), view namespace (the name of the view folder) and the view parameter.

# Making a app for AppSpot

Defining a view is slighly easy. Create a folder in the ’view’ folder in the program directory of AppSpot, give the directory the title of the namespace of the given view and give it it’s file name ”main.view” . Open a text editor, and write this:

|  |
| --- |
| Main.view |
| <vbox flex=”1” id=”mainview”>  <hbox class="sections">  <hbox class="secActive" rel="home" nmousedown="SetView('home')">  <text value="Overview" />  </hbox>  </hbox>  <deck style="" flex="1" selectedIndex="0" scrollbars="yes">  <vbox style="width:100%;overflow:auto;" flex="1" id="home" class="Section">  </vbox>  </deck>  </vbox> |
|  |

## Required Structure

There is some requirements when it comes to the structure of the XUL file for each SpotiApps.

**Header VBOX**

<vbox flex=”1” id=”mainview”>

</vbox>

The header of the page with the section buttons is defined with the header VBOX element as first declaration. This element should be assigned the ID ”mainwindow” and with flex to 1.

**Section bar**

<hbox class="sections">

<hbox class="secActive" rel="home" nmousedown="SetView('home')">

<text value="Overview" />

</hbox>

</hbox>

The top bar of each view consists of sections. Those sections is individual parts of the view that can be run independent of each other. The armature for the sections is defined as a hbox element with the class ”sections”.

**Section tabs**

The tabs for the section is defined by an additional hbox element inside the previous tag explained here and reprents the tab sections of the view. The rel attribute should match the id of the container of the section inside the deck we will explain further.

The first view (which is enabled by defualt) has the following attributes set

Class = secActive

Rel = standard

Onmousdown = SetView(’home’)

Inside this tag, a additional tag ’text’ contains the title of the sectino.

## The action deck

The deck where the section content relies on is specified by this tags:

<deck style="" flex="1" selectedIndex="0" scrollbars="yes">

[ content]

</deck>

Inside this, each section is consisting of a vbox element, whose ID is matching the rel attribute of the sections.

<vbox style="width:100%;overflow:auto;" flex="1" id="home" class="Section">

// contents goes here

</vbox>

Note that it should have the properties so to it which are shown above to get a correct experience.

**The ”rel” attribute on the corresponding section element must be equal to the vbox’s ID field.**

## Elements

Then elements can be applied to the various section. In addition to the specific themed items that are specific for AppSpot, the element system is equivalent to the XUL system. All elements in XUL are usable here and the specific elements are derived from it.

### Sections

Section element can be done by this (like those in artist view)

|  |
| --- |
|  |
| <hbox class="section">  <text class="sectionText" value="Songs"></text>  </hbox> |

## Session token

The $argv[1] variable converts to a simple string consisting of the value of the parameter specified when opening the view. It must be encapsulated into a string token , otherwise a parser error will occur.

### URI calls

As the XUL viewer component are independent of the player, I will define some URI hooks that triggers the app to do a certin task. The URI starts with ”http://go.go/” (will not browse there though) and then specify the URI that the system handles with it’s URI parser. The browser will not try to open the uri in the reality because the handler will request the XUL browser to stop.

Some notable elements:

* **A image that are set** to the class ’image’ will create a image like those in Spotify.

# Gathing feeds

Feeds for content inside a AppSpot view are defined by preprocessing instructions in the PHP core. The instructions can be done as following:

|  |
| --- |
| <?php  function createPlaylist($name,$description,$author,$img,$url){ ?>  <vbox flex="1" style="width:100%">    <hbox style="width:100%;padding:4px">  <vbox style="margin:5px">  <image class="image" src="<?php echo $img?>"/>  <box flex="1"></box>  </vbox>  <vbox>  <hbox >  <text class="header3" value="<?php echo $name?>" style="width:100%"/>  <text class="header4" value=" (totalspotify)" style="width:100%"/>  </hbox>  <description flex="1" ><?php echo str\_replace("<","<html:",$description)?></description>  </vbox>  </hbox>    </vbox>  <box class="divider" style="width:100%" height="32"></box>  <?php } ?> |
| <vbox flex="1" id="content" style="width:100%">  <?php  if(sizeof($argv)>0){  $query = $argv[1];  $a = file\_get\_contents("http://ws.spotify.com/search/1/track?q=$query");    $f = new DOMDocument();  $f->loadXML($a);  $g = $f->getElementsByTagName("item");  for($i=0; $i < $g->length - 1; $i++){  $item = $g->item($i);  $name = $item->getElementsByTagName("title")->item(0)->nodeValue;  $description =$item->getElementsByTagName("artist")->item(0)->getElementsByTagName("artist")->item(0)->nodeValue;  $url = $item->getAttribute("href");  createPlaylist($name,$description,"test","http://www.spotify.com/wp-content/themes/spotify/images/header/logo.png",$link);  }  }  ?>    </vbox> |

The first snippes define a function that can generate a set of elements according to parameters in the function createPlaylist(). The second puts its in the work in a loop that takes the third token in the view as a query to the view and handle it through the argv variable. The 1 place in the argv variable holds the argument of the current view.

The argument can store query data for requesting remote servers for getting list of contents in the view from external sources.

# Javascript

As usual, the XUL views

## XMLHttp

Unfortunally, the XUL component does not allow external requests from XMLHTTP in scripts running on the XUL pages. However, we have done a workaround where a fake URI is handled by the client application and then sends it to a file which are opned by a special hook.

So feeds are done in this way

|  |
| --- |
| Localscript.js |
| Function callFromElement(){  HttpRequest(”http://someuri//feed.xml”,”feed”,whenready);  }  Function whenready(e){  // e is a instance of document.implementation.createDocument() which loads the hooked request from a file stored by the host application. Access XML content by e.documentElement or text data by e.documentElement.firstChild.nodeValue  } |
|  |