appMobi {!}

Native Mobile Application Development Ecosystem

API Documentation

Build 3.2   10/19/2010

Table of Contents

1.0 PURPOSE

The purpose of this document is to describe to application programmers the prerequisites and APIs for writing an application using the appMobi ecosystem.

This document should allow a web designer with a working knowledge of JavaScript and HTML to create a mobile application for a variety of platforms such as the iPhone, iPad, or Android devices.

2.0 PROGRAMMING PREREQUISITES

- You must have the appMobi XDK development environment as well as an appMobi developer account in order to develop apps. Download the XDK and sign up for your free account at:
-

  http://xdk.appmobi.com

- In order to test your application while you are developing it, you'll need "test container" software running on your device. If you don't have a test container yet, you can install one by pointing your mobile device's web browser to:

  http://www.appmobi.com/mobile

3.0 APPMOBI BEST PRACTICES

Every appMobi app is essentially a miniature web site that is deployed directly to the target device. However, in order to function correctly the following stipulations apply:

- The application mini-website may include a subdirectory named "*_appMobi*" in the root of its file structure. This directory is used to hold any custom images such as the application's splash image while it starts up or for skinning the native player screen. The "*_appMobi*" directory is not mandatory to make a functional application.

- Every page that uses the appMobi API commands on the mini-site must reference "*http://localhost:58888/_appMobi/appmobi.js*" through a script tag. For example in the head tag of the page it should include this line of HTML:

```
<script type="text/javascript"
src="http://localhost:58888/_appMobi/appmobi.js"></script>
```

- Every page that needs to access the appMobi API commands (like accelerometer etc.) must wait for the "*appMobi.device.ready*" event to fire before making any requests.

- If your app has subdirectories in its file structure, you can use relative paths to refer to the files, not root relative paths. For example, *"../xxx.html"* or "*yyy.html*" are good but *"/mydirectory/page.html*" will not work. You can also use the local webserver to reference your pages. The root directory is accessible at the root of the web server at "*http://localhost:58888/*". This means that the following are equivalent:

  ```
  <img src=”logo.gif”/>
  ```

  and

  ```
  <img src=”http://localhost:58888/logo.gif”/>
  ```

- The initial page of your app is always "*index.html*". If other pages link back to it, the name must always be lowercase only.

- Please be aware that the test container is not able to open new browser windows as a conventional web browser would. For example, using the window.open JavaScript command will start a new instance of Safari on an iOS device such as the iPhone or iPad.

3.1 Register Device Ready

Before your app makes any calls to the appMobi library, the library must be ready. Add an event listener for the *appMobi.device.ready* event to determine when the appMobi library is loaded.

```
<head>
<script type="text/javascript" charset="utf-8"
src=”http://localhost:58888/_appMobi/appmobi.js” />
<script language="javascript" type="text/javascript">
        function onDeviceReady()
        {
            ...
        }
         //put the event listener after the device handler
        document.addEventListener("appMobi.device.ready",onDeviceReady,false);
</script>
</head>
```

4.0 APPMOBI LIBRARY

4.1 Object Diagram

The top level *AppMobi* object encapsulates all the functionality of the appMobi library. It includes all objects below it, as well as a single parameter that holds the version of the Javascript library.

- ❖ AppMobi
  - ➢ jsVersion
  - ➢ accelerometer
  - ➢ cache
  - ➢ device
  - ➢ display
  - ➢ geolocation
  - ➢ notification
  - ➢ player
  - ➢ playingtrack
  - ➢ stats

4.2 jsVersion Property

This property holds the current version of the appMobi Javascript library used in the application. It can be compared to the appMobi application version stored in *AppMobi.device.appmobiversion*.

Example:
```
alert("AppMobi Javascript Library Version: " + AppMobi.jsVersion);
```

4.3 accelerometer Command Object

The *accelerometer* object is used to track the accelerometer on the device.

- Success and failure callback functions need to be defined in your Javascript.
- Successful data is returned as an object with the attributes ".x", ".y", and ".z".
- Values of accelerometer samples for each axis range from -1 to 1.

4.3.1 watchAcceleration([success function],[acceleration options object])

This method will asynchronously acquire the acceleration repeatedly at a given interval.

The *acceleration options* parameter is looking for a javascript object with two properties: ".frequency" which changes that millisecond interval for refresh and has a default of 500 and ".adjustForRotation" which changes the values of the .x and .y parameter based on the device orientation. It is defined in the code as *AppMobi.AccelerationOptions*.

The success function callback is triggered each time data is available. Successful data is returned as an object with the attributes ".x", ".y", and ".z".

This method returns a *watch timer* object to be cleared with *clearWatch.*

4.3.2 clearWatch([watch timer object])

This method stops the process started by *watchAcceleration* when it is passed the appropriate *watch timer* object.

Example:
```
AppMobi.acceleremoter.clearWatch(watchId);
```

4.3.3 Code Sample for Accelerometer

For added clarity, an extended sample of how to use the accelerometer is included below.

```
<html>
<head>

<script type="text/javascript" charset="utf-8"
src="http://localhost:58888/_appMobi/appmobi.js"/>
<script language="javascript">

//define a global variable that will contain the interval timer for accelerometer
var timer;

//use this to round a sample to 3 decimals of precision
function roundNumber(num)
{
    var dec = 3;
    var result = Math.round(num*Math.pow(10,dec))/Math.pow(10,dec);
    return result;
}

var watchAccel = function()
{
       //provide a success function – defining it inside the watchAccel
       //function is a nicety that hides its scope. You could definite
       //it globally just as well.

       var suc = function(a)
       {

       //readings are from -1 to 1 (with 0 being equilibrium in a
       //plane). Assumes holding in portrait mode
       //with screen pointed straight at your chest.
       //e.g. in the X plane -1 = tilted all the way left, 1 = tilted all the way right.
       //e.g. in the Y plane -1 = tilted all the way left, 1 = tilted all the way right.
       //use our rounding function to get a value, that print that value to an output div
       //so you can see the results.

       document.getElementById('accel_x').innerHTML = roundNumber(a.x);
       document.getElementById('accel_y').innerHTML = roundNumber(a.y);
       document.getElementById('accel_z').innerHTML = roundNumber(a.z);

       };
```

```
        //create the options object and set the frequency to receive samples
        var opt = new AppMobi.AccelerationOptions();
        opt.frequency = 100;

        //use the special timer variable which will send the samples back
        timer = AppMobi.accelerometer.watchAcceleration(suc,opt);

}


/* When this function is called, appMobi has been initialized and is ready to roll */
function onDeviceReady()
{
        watchAccel();
}
document.addEventListener("appMobi.device.ready",onDeviceReady,false);


</script>
</head>

<body onload="onBodyLoad()"><dl id="accel-data">
      <dt>X:</dt><dd id="accel_x">&nbsp;</dd>
      <dt>Y:</dt><dd id="accel_y">&nbsp;</dd>
      <dt>Z:</dt><dd id="accel_z">&nbsp;</dd></dl>
</body>
</html>
```

4.4 cache Object

This object is intended to provide local storage for data to speed up applications. It can be used as in conjunction with, or as an alternative to the HTML5 local database. Its methods provide features similar to browser cookies and file caching.

For cookies, the intention is that you would use *setCookie* to save string data in name-value pairs. Cookies persist between application sessions.
Data values may be retrieved using the *getCookie* command or from the *getCookieList* command as soon as the "*appMobi.device.ready"* event is fired.

The media cache commands are meant to provide quicker access to files such as videos and images. Adding files to the media cache will expedite access to them when the application runs. These are files that are cached across sessions and are not in your application bundle. See the section on events below for more information about events fired from the cache section of appMobi.

4.4.1 setCookie([name],[value],[expirationDays])

Call this method to set a chunk of data that will persist from session to session. The data is automatically purged once the expiration date lapses. The data can be retrieved using the *getCookie* command.

Please note that cookie names may not include periods.

Example:
```
function saveInfo() {
        //add a cookie
        var name = prompt('Enter information name:');
        var value = prompt('Enter information value:');
        var daysTillExpiry = prompt('Days until cookie expires (-1 for never):');
        try
        {
                if (name.indexOf('.')!= -1)
                {
                        AppMobi.cache.setCookie(name,value,daysTillExpiry);
                }
                else
                {
                        alert('cookie names may not include a period');
                }
        }
        catch(e)
        {
                alert("error in saveInfo: " + e.message);
        }
}
```

4.4.2 getCookie([name])

This method will get the value of a cookie previously saved using the *setCookie* command. If no such cookie exists, the value returned will be "undefined".

Example:

```
var value = AppMobi.cache.getCookie('userid');
```

4.4.3 getCookieList()

This method will return an array containing all the names of all the previously saved cookies using the *setCookie* command. These names can then be used in calls to *getCookie*.

Example:

```
var cookiesArray = AppMobi.cache.getCookieList();
for  (var x=0;x<cookiesArray.length;x++)
{
  alert( cookiesArray[x] + "   " + AppMobi.cache.getCookie(cookiesArray[x])  );
}
```

4.4.4 removeCookie([name])

This method will clear a cookie previously saved using *setCookie*.

Example:

```
AppMobi.cache.clearCookie('userid');
```

4.4.5 clearAllCookies()

This method will clear all cookies.

Example:

```
AppMobi.cache.clearAllCookies();
```

4.4.6 addToMediaCache([url])

This command will get a file from the Internet and cache it locally on the device. It can then be referenced in a special directory named *_mediacache* off the root of the bundle. Once this command is run, the "*appMobi.cache.media.add"* event is fired. If there is already a file cached with that name it is overwritten.

Example:
```
AppMobi.cache.addToMediaCache(urlToCache);

function cacheUpdated(e)
{
       alert(e.url + " cached successfully");
}
document.addEventListener('appMobi.cache.media.add', cacheUpdated, false);
```

4.4.7 removeFromMediaCache([url])

This command will remove a file from the local cache on the device. Once this command is run the "*appMobi.cache.media.remove"* event is fired.

Example:
```
AppMobi.cache.removeFromMediaCache(urlToRemove);

function cacheUpdated(e)
{
       alert(e.url + " removed successfully");
}
document.addEventListener('appMobi.cache.media.remove', cacheUpdated, false);
```

4.4.8 clearMediaCache()

This command will remove all files from the local cache on the device. Once this command is run the "*appMobi.cache.media.clear"* event is fired.

Example:
```
AppMobi.cache.clearMediaCache();

function cacheCleared()
{
       alert("cache cleared successfully");
}

document.addEventListener('appMobi.cache.media.clear', cacheCleared, false);
```

4.4.9 getMediaCacheList()

This method will get an array containing all the names of all the previously cached files using the *addToMediaCache* command. These names can then be used in calls to *getMediaCacheLocalURL*.

Example:
```
var cacheArray = AppMobi.cache.getMediaCacheList();
for (var x=0;x<cacheArray.length;x++)
{
 alert( cacheArray[x] + "  " + AppMobi.cache.getMediaCacheLocalURL(cacheArray[x]) );
}
```

4.4.10 getMediaCacheLocalURL([url])

This method will return an url that you can use to access the cached media file. If it is not cached, the value returned will be “undefined”.

Example:
```
var localurl = AppMobi.cache.getMediaCacheLocalURL(‘http://myweb.com/image/logo.gif’);
```

4.5 device Object

The *device* object provides access to information about the device itself through a series of properties and functions. See the events section of this document below for more information on events thrown by the device object.

4.5.1 platform

The *platform* property returns a text string identifying the platform that appMobi is running on. Valid values include:

Platform	**Information**
iOS	This platform is used on Apple’s iPhone, iPod, and iPad devices.
Android	This platform is used on any device that runs the Android operating system.

Example:
```
alert(AppMobi.device.platform);
```

4.5.2 model

This property returns the model of the device that the application is running on.

Example:
```
alert(AppMobi.device.model);
```

4.5.3 uuid

The *uuid* property returns the device’s unique identification id.

Example:
```
alert(AppMobi.device.uuid);
```

4.5.4 osversion

This property returns the device’s current operating system version information.

Example:
```
alert(AppMobi.device.osversion);
```

4.5.5 appmobiversion

This property returns the version of the appMobi container software that the application is using.

Example:
```
alert(AppMobi.device.appmobiversion);
```

4.5.6 initialOrientation

This property returns the current orientation of the device. It will return one of the following values:

Orientation	**Value**
Portrait	0
Upside Down Portrait	180
Landscape Right	90
Landscape Left	-90

Example:
```
//detect the initial orientation of the device
if (AppMoibi.device.initialOrientation == "90" || AppMobi.device.initialOrientation == "-
90")
{
        //landscape
}
else
{
        //portrait
}
```

4.5.7 phonegapversion

This property returns the version of phonegap running below the *AppMobi* layer.

Example:
```
alert(AppMobi.device.phonegapversion);
```

4.5.8 density

For future use.

4.5.9 hasAnalytics

This property says whether analytics has been enabled for this application. Functions under AppMobi.stats.* will not be available if this is false.

Example:
```
alert(AppMobi.device.hasAnalytics);
```

4.5.10 hasCaching

This property says whether caching has been enabled for this application. Functions under AppMobi.cache for mediacache will not be available if this is false.

Example:
```
alert(AppMobi.device.hasCaching);
```

4.5.11 hasStreaming

This property says whether streaming has been enabled for this application. Functions under AppMobi.player for station and shoutcast will not be available if this is false.

Example:
```
alert(AppMobi.device.hasStreaming);
```

4.5.12 hasAdvertising

For future use

4.5.13 lastStation

This property will hold the *NetStationID* or ShoutcastURL of the station that is playing. Otherwise, this property only holds a null. This allows the user to know if their station is already playing at startup. This can happened if the UI is destroyed and the audio continues in the background and then the application is later brought back to the foreground.

Example:
```
alert(AppMobi.device.lastStation);
```

4.5.14 connection

This property returns the best type of internet connection available and returns the result. This property is updated only when the *device.updateConnection()* command is called and the *appMobi.device.connection.update* event is triggered.

Connection	**Information**
wifi	The device has an active wifi connection
cell	The device has an active cellular connection
none	The device does not currently have an Internet connection

Example:
```
document.addEventListener(“appMobi.device.connection.update”,function(){alert(AppMobi.dev
ice.connection);},false);

AppMobi.device.updateConnection();
```

4.5.15 managePower([shouldStayOn],[onlyWhenPluggedIn])

This function controls how the device behaves in certain power states. It is passed two Boolean values. If *shouldStayOn* is false, normal power management for the device applies. If *shouldStayOn* is true and *onlywhenPluggedIn* is true, then the device will not go to sleep if the device is plugged in. If *shouldStayOn* is true and *onlyWhenPluggedIn* is false, then the device will never go to sleep.

Example:
```
AppMobi.device.managePower(true,false);
```

4.5.16 setAutoRotate([shouldAutoRotate])

This function will control whether the device should automatically handle rotation or not based on a boolean value.

Example:
```
AppMobi.device.setAutoRotate(false);
```

4.5.17 setRotateOrientation([orientation])

This function will lock the orientation of the device to either “landscape” or “portrait” orientation. The orientation will be locked based on which specific string is passed to the function. Passing “portrait” will lock the application into portrait orientation and passing “landscape” will lock the application into landscape orientation. If the current orientation is not the specified orientation, the device will lock in the specified orientation only once the device is oriented in that position.

To unlock the orientation, set the orientation to an empty string.

Example:
```
//lock in "portrait" orientation
AppMobi.device.setRotateOrientation("portrait");
```

4.5.18 updateConnection()

This function will query the device to determine its current connection to the internet. When it is done it will fire an event “appMobi.device.connection.update” and the connection property of AppMobi.device will be updated.

Example:
```
AppMobi.device.updateConnection();
```

4.5.19 setBasicAuthentication([domain], [username], [password])

This function will set header data required for basic authentication over the Internet. It requires the domain and realm of the server it will make the request to as well as the appropriate username and password credentials. This method is not available on all platforms.

Example:
```
AppMobi.device.setBasicAuthentication('api.twitter.com', username, password);

try
{
    xmlhttp = new XMLHttpRequest(); // instantiate XMLHttpRequest
}
catch (err)
{
    alert("Error initializing XMLHttpRequest.\n"+ err);
    return;
}

xmlhttp.onreadystatechange = function()
{
    alert(xmlhttp.status + "   " + xmlhttp.readyState);
    if (xmlhttp.readyState == 4)
    {
        if(xmlhttp.status == 200)
        {
            try { RequestResponse(url,true,xmlhttp.responseText); } catch(e) {}
        }

    }
}

try
{
   xmlhttp.open('GET', url);
}
catch (err)
{
    alert("XMLHttpRequest.open() failed.\n"+err.message + " \n URL : " + url);
    return;
}

xmlhttp.send(strData);
```

4.5.20 addVirtualPage()

This function will intercept a single press of the device's hardware "back" button and fire the "*appMobi.device.hardware.back"* event instead. Call this function several times to intercept multiple device hardware button presses. This is used in applications that want to simulate flow through their app and have the device's hardware back button be able to be used to navigate backwards through the flow.

This method is not available on all platforms.

Example:
```
AppMobi.device.addVirtualPage();
```

4.5.21 removeVirtualPage()

This function will remove the interception of a single press of the device's hardware "back" button. This method is not available on all platforms.

Example:
```
AppMobi.device.removeVirtualPage();
```

4.5.22 launchExternal([url])

This function will open an url in an external browser window. For example, on an iOS device, the URL will open in a Safari window.

Example:
```
AppMobi.device.launchExternal("http://www.google.com");
```

4.5.23 showRemoteSite([url],[closeImageX],[closeImageY],[closeImageWidth],[closeImageHeight])

This function is used to show a remote web site in a different web view. Touching the close image will shut down the web view and return the user to the normal appMobi application.

The *url* parameter is for the new view's target url. The image coordinates define the position, width, and height of the close image that the user may touch to close the web view. By default close image is set to 48x48 pixels and positioned in the upper left hand corner of the screen.

The close image may be changed by including a different *remote_close.png* file in the _appMobi directory of the project.

When the close button is touched, it fires an *appMobi.device.remote.close* event.

Example:
```
//Display the Lancaster Day Care Center site to learn more
AppMobi.device.showRemoteSite("http://www.ldcc.org/",280,0,50,50);
```

Version: This command is available in appMobi 3.2.

4.5.24 getRemoteData([url],[requestMethod],[requestBody],[successCallback],[errorCallback])

This function is used for making background POST/GET requests of XML data. It is an alternative to the HTML XMLHttpRequest function.

The *url* parameter should hold the URL to request the XML data from. The *requestMethod* must be either "GET" or "POST". The *requestBody* is unused for a request with the "GET" method (just pass an empty string) and should hold the post data for a "POST" method request in a name=value format separated by ampersands. The *successCallback* should hold a function with a single parameter holding the data returned from the success. The *errorCallback* should hold a similar function with a single parameter holding the data returned from the error.

Example:
```
//GET method example
AppMobi.device.getRemoteData("http://ldcc.broadp3.com/auction/index.php?REQUEST=GET","GET
","","success_handler","error_handler");

//POST method example
AppMobi.device.getRemoteData("http://twitter.com/statuses/public_timeline.xml","POST","E-
MAIL=support@appmobi.com&TEST=1&MAX=0"","success_handler","error_handler");

//Example Event Handlers
function success_handler (data) {  alert("success: " + data); }
function error_handler(data) {  alert("error: " + data); }
```

Version: This command is available in appMobi 3.2.

4.6 display Object

The *display* object gives the application control over aspects of the device's video display properties.

4.6.1 useViewport([widthPortrait], [widthLandscape])

This method tells appMobi to size down an application for a smaller device. Use this method in your application's *appMobi.device.ready* event to size a larger application down for use on a smaller device than it was originally designed for.

This method expects two values, the first is the width the application is designed for in portrait orientation, and the second is the width the application is designed for in landscape orientation.

Example:
```
//use AppMobi viewport to allow this iPad-designed application to size down to an iPhone
//or an Android handset device with a resolution lower than 1024x768.
var iPortraitWidth=768;
var iLandscapeWidth=1024;
AppMobi.display.useViewport(iPortraitWidth,iLandscapeWidth);
```

4.6.2 startAR()

This method triggers appMobi's augmented reality mode. In this mode of operation, the application's background will show the camera input. In order to take advantage of this setting, the application's <*body*> tag must include the CSS style "*background-color:transparent*".

Please note that augmented reality mode is not available on all platforms and devices due to hardware requirements.

Augmented reality mode will lock to a particular orientation at startup.

Example:
```
AppMobi.display.startAR();
```

4.6.3 stopAR()

This method turns off augmented reality mode.

Example:
```
AppMobi.display.stopAR();
```

4.7 geolocation Object

4.7.1 getCurrentPosition([success function], [error function])

Use this command to get the current location. This command asynchronously acquires the approximate latitude and longitude of the device. When data is available, the success function is called. If there is an error getting position data, the error function is called.

Example:
```
var getLocation = function()
{
    var suc = function(p){
        alert("geolocation success");
        if (p.coords.latitude != undefined)
        {
            currentLatitude = p.coords.latitude;
            currentLongitude = p.coords.longitude;
        }

    };
    var fail = function(){
        alert("geolocation failed");
        getLocation();
    };

    AppMobi.geolocation.getCurrentPosition(suc,fail);
}
```

4.7.2 watchPosition([success function], [error function],[options])

Use this command rather than the getCurrentPosition command to track progress during a trip rather than just getting a single position. This command asynchronously acquires the latitude and longitude of the device. When data is available, the success function is called. If there is an error getting position data, the error function is called.

Some options may be specified for the function as an array object.

Option	Possible Values	Use
timeout	[a number]	The number of milliseconds between checks of position rather than the default value of 10000 (or ten seconds).
enableHighAccuracy	True	This will force the command to report back a more accurate

		latitude and longitude position at the expense of more battery usage
maximumAge	[a number]	The number of milliseconds the command will wait before deciding that it cannot get a new position reading and instead run the error message

This function will return a *watch timer* object that can be used to stop the watch using *AppMobi.geolocation.clearWatch*.

Example:
```
//This array holds the options for the command
var options = {timeout: 10000, maximumAge: 11000, enableHighAccuracy: true };

//This function is called on every iteration of the watch Position command that fails
var fail = function(){
  alert("Geolocation failed. \nPlease enable GPS in Settings.");
};

//This function is called on every iteration of the watchPosition command that is a
success
var suc = function(p){
  alert(“Moved To: Latitude:” + p.coords.latitude + “Longitude: “ + p.coords.longitude;
};

//This command starts watching the geolocation
var geolocationWatchTimer = AppMobi.geolocation.watchPosition(suc,fail,options);

//Call the stopGeolocation function to stop the geolocation watch
var stopGeolocation = function(){
       AppMobi.geolocation.clearWatch(geolocationWatchTimer);
}
```

4.7.3 clearWatch([watch timer])

This method stops the process started by *watchPosition* when it is passed the appropriate *watch timer* object.

Example:
```
var geolocationWatchTimer = AppMobi.geolocation.watchPosition(suc,fail,options);
AppMobi.geolocation.clearWatch(geolocationWatchTimer);
```

4.8 notification Object

The *notification* object allows the developer to alert the user using device-specific capabilities.

4.8.1 beep([count])

This method will force the device to beep. Passing a numeric value will cause it to beep several times in succession. If no parameters are passed, the number of beeps defaults to 1.

Example:
```
AppMobi.notification.beep(1);
```

4.8.2 vibrate()

This method will force the device to vibrate.

Example:
```
AppMobi.notification.vibrate();
```

4.8.3 alert([message],[title],[buttontext])

This method will display a modal alert box. The message, alert box title, and the text on the confirm button are all defined by the parameters passed to this method.

Example:
```
AppMobi.notification.alert("Hammertime!","STOP","Can\'t Touch This");
```

4.8.4 showBusyIndicator()

This method will turn on the device's "working" or "busy" state graphics such as a spinner or an hourglass. Turn it off using the *hideBusyIndicator* method.

Example:
```
AppMobi.notification.showBusyIndicator();
```

4.8.5 hideBusyIndicator ()

This method will turn off the device's "working" or "busy" state graphics such as a spinner or an hourglass. Turn it on using the *showBusyIndicator* method.

Example:
```
AppMobi.notification.hideBusyIndicator();
```

4.8.6 getNotificationList()

Use this method to get a list of id keys to obtain access to the list of available notifications for this user. It returns a javascript array of keys. This array is valid until an *appMobi.notifiation.push.receive* or *appMobi.notification.push.refresh* event fires or if *readPushNotifications* is called.

Example:
```
var  myNotifications=AppMobi.notification.getNotificationList();
var len=myNotifications.length;
if(len>0){

    var strMessages="";
    for(i=0;i<len;i++)
    {
        msgObj=AppMobi.notification.getNotificationData(myNotifications[i]);
        try{
            if(typeof  msgObj=="object"&&msgObj.id==myNotifications[i]){
            strMessages+=unescape(msgObj.msg)+"<br>\n";
            clearList.push(msgObj.id);
            }
        }catch(e){alert("Invalid message object");}
    }
    alert(strMessages);
}
```

Version: This command is available in appMobi 3.2.

4.8.7 getNotificationData([id])

Use this method to get any data associated with a notification. This method requires the id key of a particular notification. The id keys can be obtained through the *getNotificationList* command. This method will return a null if there is no notification for the specified id key found. On a success, it will return an object with the properties “id”, “msg”, and “data”. The “id” property will hold the notification’s id key, the “msg” property will hold the notification’s message, and the “data” key will hold any data sent along with the notification.

Example:

```
var  myNotifications=AppMobi.notification.getNotificationList();
var len=myNotifications.length;
if(len>0){

    var strMessages="";
    for(i=0;i<len;i++)
    {
        msgObj=AppMobi.notification.getNotificationData(myNotifications[i]);
        try{
            if(typeof  msgObj=="object"&&msgObj.id==myNotifications[i]){
            strMessages+=unescape(msgObj.msg)+"<br>\n";
            clearList.push(msgObj.id);
            }
        }catch(e){alert("Invalid message object");}
    }
    alert(strMessages);
}
```

Version: This command is available in appMobi 3.2.

4.8.8 checkPushUser([userID],[password])

Use this method to confirm the credentials of a user or to log an existing user in on a different device. This method requires two parameters. The first parameter requires a unique string (per appMobi application) for this user to be addressable in the push system. The second parameter required is the user’s password for the push system.

Following a call to this method, it will fire the *appMobi.notification.push.enable* event that includes a “success” property that is set to true or false. On an error, this event will additionally include a “message” property describing the error.

Example:

```
document.addEventListener("appMobi.notification.push.enable",notificationsRegistered,fals
e);
AppMobi.notification.checkPushUser("userID","newpassword");
var notificationsRegistered=function(event)
{
       if(event.success===false)
```

```
    {
       if( event.message=="user already exists" )
       {
           AppMobi.notification.checkPushUser(“userID”,"newpassword");
           didcheckuser = true;
        }
        else
        {
              alert("There was an error adding push notifications "+event.message);
        }
        return;
    }
       alert("Notifications Enabled");
}
```

Version: This command is available in appMobi 3.2.

4.8.9 addPushUser([userID],[password],[email])

Use this method to register a new user id on a particular appMobi application with the appMobi push service. This method requires three parameters. The first parameter requires a unique string (per appMobi application) for this user to be addressable in the push system. The second parameter required is the user’s chosen password for the push system. The third parameter is the email address that the *sendPushUserPass* function will use when a user asks to retrieve their password.

Following a call to this method, it will fire the *appMobi.notification.push.enable* event that includes a “success” property that is set to true or false. On an error, this event will additionally include a “message” property describing the error.

Example:

```
document.addEventListener("appMobi.notification.push.enable",notificationsRegistered,fals
e);
AppMobi.notification.addPushUser("userID","newpassword","newEmail@test.com");
var notificationsRegistered=function(event)
{
       if(event.success===false)
    {
       if( event.message=="user already exists" )
       {
           AppMobi.notification.checkPushUser(“userID”,"newpassword");
           didcheckuser = true;
        }
        else
        {
              alert("There was an error adding push notifications "+ event.message);
        }
        return;
    }
       alert("Notifications Enabled");
}
```

Version: This command is available in appMobi 3.2.

4.8.10 editPushUser([newEmail],[newPassword])

Use this method to change the email address and the password associated with the push notification service of an appMobi application. This method requires two parameters. The first parameter is the email address that the *sendPushUserPass* function will use when a user asks to retrieve their password. The second parameter required is the user's chosen password for the push system. To leave either parameter as it was, simply pass an empty string instead.

Following a call to this method, it will fire the *appMobi.notification.push.user.edit* event that includes a "success" property that is set to true or false. On an error, this event will additionally include a "message" property describing the error.

Example:
```
document.addEventListener("appMobi.notification.push.user.edit",updateNotificationEvent,f
alse);
var updateNotificationEvent=function(event)
{
       if(event.success==false)
    {
       alert("error: " + event.message)
    }
       else
       {
           alert("success");
       }
}
AppMobi.notification.editPushUser("test@appmobi.com","newpassword");
```

Version: This command is available in appMobi 3.2.

4.8.11 sendPushUserPass()

Use this method to send an email message to a user including the password for push notifications for a particular appMobi application.

Following a call to this method, it will fire the *appMobi.notification.push.sendpassword* event that includes a "success" property that is set to true or false. On an error, this event will additionally include a "message" property describing the error.

Example:
```
document.addEventListener("appMobi.notification.push.user.edit",updateNotificationEvent,f
alse);
var updateNotificationEvent=function(event)
{
       if(event.success==false)
    {
       alert("error: " + event.message)
    }
       else
       {
           alert("success");
       }
}
AppMobi.notification.sendPushUserPass();
```

Version: This command is available in appMobi 3.2.

4.8.12 setPushUserAttributes([homeLatitude],[homeLongitude],[currentLatitude],[currentLongitude],[birthYear],[gender])

Use this method to associate demographic information for a user. This data will be used on the appMobi Push Notifications Service Admin Control Panel to allow applications to target push notifications to particular users of an appMobi application.

This method requires several parameters.

Parameter	Explanation
homeLatitude	The latitude for the user's home
homeLongitude	The longitude for the user's home
currentLatitude	The user's current Latitude
currentLongitude	The user's current Longitude
birthYear	The user's birth year
gender	The user's gender

Following a call to this method, it will fire the *appMobi.notification.push.user.editattributes* event that includes a "success" property that is set to true or false. On an error, this event will additionally include a "message" property describing the error.

Example:
```
AppMobi.notification.setPushUserAttributes(-40.3,-70.1,31.3,64.1,28,"m");
document.addEventListener("appMobi.notification.push.user.editattributes",updateNotificat
ionEvent,false);
var updateNotificationEvent=function(event)
{
        if(event.success==false)
    {
        alert("error: " + event.message)
    }
        else
        {
            alert("success");
        }
}
```

Version: This command is available in appMobi 3.2.

4.8.13 findPushUser([userID],[email])

Use this method to allow users to find other users of an appMobi application. It provides a kind of "friends" functionality.

This method requires two parameters. The first parameter is a unique user id to look for in the messaging database of the application. The second parameter is an email address to look for in the messaging database of the application. One or both parameter may be specified. Pass an empty string to one parameter to search only by the other. Passing both parameters will default to a search on the user id. If both parameters are sent as an empty string, a random users will be selected from the messaging database.

Following a call to this method, it will fire the *appMobi.notification.push.user.find* event that includes a “success” property that is set to true or false. On an error, this event will additionally include a “message” property describing the error. On a success, this event will additionally include a “userid” property that contains a matching user id.

Example:
```
AppMobi.notification.findPushUser("testUserID","TestUser@appmobi.com");
document.addEventListener("appMobi.notification.push.user.editattributes",updateNotificat
ionEvent,false);
var updateNotificationEvent=function(event)
{
        if(event.success==false)
    {
        alert("error: " + event.message)
    }
        else
        {
            alert("success");
        }
}
```

Version: This command is available in appMobi 3.2.

4.8.14 refreshPushNotifications()

Use this method to allow users to force a user’s device to re-synchronize notifications between this device and the database on the server.

Generally this function is not needed, but this method is featured to force fresh data to be returned.

Following a call to this method, it will fire the *appMobi.notification.push.refresh* event that includes a “success” property that is set to true or false. On an error, this event will additionally include a “message” property describing the error.

Example:
```
AppMobi.notification.refreshPushNotifications();
document.addEventListener("appMobi.notification.push.user.refresh",updateNotificationEven
t,false);
var updateNotificationEvent=function(event)
{
        if(event.success==false)
    {
        alert("error: " + event.message)
    }
        else
        {
            alert("success");
        }
}
```

Version: This command is available in appMobi 3.2.

4.8.15 readPushNotifications([notificationIDs])

Use this method to remove notifications from the server.

This method requires a single parameter, a pipe ("|") delimited list of notification ids to mark as read and remove from the system.

Following a call to this method, it will fire the *appMobi.notification.push.read* event that includes a "success" property that is set to true or false. On an error, this event will additionally include a "message" property describing the error.

Example:
```
//clearList is an array of notification ids
if(clearList.length>0) {
 AppMobi.notification.readPushNotifications(clearList.join("|"));
}

document.addEventListener("appMobi.notification.push.user.refresh",updateNotificationEven
t,false);
var updateNotificationEvent=function(event)
{
        if(event.success==false)
    {
        alert("error: " + event.message)
    }
        else
        {
            alert("success");
        }
}
```

Version: This command is available in appMobi 3.2.

4.8.16 sendPushNotification([userID],[message],[data])

Use this method to send a push notification to another user of an application.

This method requires three parameters. The first parameter is the user id of the user to send the message to, the second parameter is the message text to send, and the third parameter is any extra application-specific data that should be attached to the notification. The third parameter can be anything and is simply passed along to the receiving application.

Following a call to this method, it will fire the *appMobi.notification.push.send* event that includes a "success" property that is set to true or false. On an error, this event will additionally include a "message" property describing the error.

Example:
```
//myUserID is a variable that holds the id to send the message to
//msg is a variable that holds the message text
AppMobi.notification.sendPushNotification(myUserID,msg,{});
document.addEventListener("appMobi.notification.push.send",updateNotificationEvent,false)
;
var updateNotificationEvent=function(event)
```

```
{
        if(event.success==false)
    {
        alert("error: " + event.message)
    }
        else
        {
            alert("success");
        }
}
```

Version: This command is available in appMobi 3.2.

4.9 player Object

The *player* object is used to play media natively in applications. It is a useful alternative to the HTML5 <video> and <audio> tags. It also allows applications to play streaming and shoutcast stations with a skinnable native UI. See the chapter on events below for more information on the events fired by this object.

4.9.1 startAudio([relativeurl])

This method will load and start playing a specified audio file without a UI of any sort. It is useful for adding a response to an application event or for playing a background audio file while the user performs other actions. This method requires the audio file to be included in the application zip file, or loaded into the *_mediacache* directory using the *AppMobi.cache.addToMediaCache* command first.

Example:
```
// start playing an audio file
AppMobi.player.startAudio("sounds/cowbell.wav");
```

4.9.2 toggleAudio()

This method will pause or play the audio started by a call to *startAudio.*

Example:
```
// toggle playback of the audio
AppMobi.player.toggleAudio();
```

4.9.3 stopAudio()

This method will stop the audio started by a call to *startAudio.*

Example:
```
// stop playback of the audio
AppMobi.player.stopAudio();
```

4.9.4 startStation([NetStationID], [ResumeMode], [ShowPlayer])

This method will load a station hosted by appMobi and identified by a unique *NetStationID* code. Optionally, a boolean parameter specifying whether the station should attempt to resume from a previous play point may be set as well as a boolean value specifying whether to show the native player or not once the station is loaded are available. *ResumeMode* may not be available on all platforms.

Example:
```
AppMobi.player.startStation(“223020”,true,true);  //KOIT San Francisco
```

4.9.5 startShoutcast([ShoutcastURL],[ShowPlayer])

This method will load a station hosted by Shoutcast. The raw URL of the Shoutcast stream is the first parameter of the load command. Be sure to include the server and port number in your URL request.

A boolean parameter specifying whether to show the native player or not once the station is loaded is available.

Particular devices are only capable of handling particular types of streams. See the chapter on supported file types below for more information.

Example:
```
// EYE97 - Commercial Free Non Stop Hits of the 70's, 80's, 90's and Today
AppMobi.player.startShoutcast(“http://209.9.238.10:8008/”,true);
```

4.9.6 playPodcast([url])

This method will load a specified podcast from the web into a native player with UI. Valid video formats depend on the device that appMobi is running on. See the chapter on supported file types below.

Although it can’t play files directly from the local bundle, you can use the appMobi local webserver and reference the files using *http://localhost:58888*.

Examples:
```
//load the video podcast
AppMobi.player.playPodcast("http://blip.tv/file/get/Unboomed-DanceAcrossAmerica550.mov");

//load a video from the root of the local bundle
AppMobi.player.playPodcast("http://localhost:58888/lovely.mov");
```

4.9.7 playSound([url])

This method will play a sound with no UI or events and is not controllable. This is intended as a simple way to play sound effects in your application. This requires the sound file to be included within the application zip file

Example:
```
// play the sound
AppMobi.player.playSound("sounds/boing.wav");
```

4.9.8 play()

This method will restart playback of a paused player. This method will do nothing when called before the player is loaded. This method is used for stations started with *startStation* or *startShoutcast.*

Example:
```
AppMobi.player.play();
```

4.9.9 pause()

This method will pause the playback of a loaded player. This method will do nothing when called before the player is loaded. This method is only used for stations started with *startStation* or *startShoutcast.*

Example:
```
AppMobi.player.pause();
```

4.9.10 stop()

This method will stop playback of a loaded player. This method will do nothing when called before the player is loaded. This method is only used for stations started with *startStation* or *startShoutcast.*

Example:
```
AppMobi.player.stop();
```

4.9.11 volume([volumePercentage])

This method controls the volume of streaming audio. It does not change the volume of the device or of any other sounds or audio played. It only allows for adjustment of the currently playing stream. This method is only used for stations started with *startStation* or *startShoutcast.*

Example:
```
AppMobi.player.volume(90);  //crank up the volume to 90%
```

4.9.12 rewind()

This method will rewind the play head to the beginning of the track or 30 seconds from its current position, whichever is less. This method will do nothing when called before the player is loaded. This method is only used for stations started with *startStation* or *startShoutcast.*

Example:
```
AppMobi.player.rewind();  //go back 30 seconds
```

4.9.13 ffwd()

This method will advance the play head to the end of the track or 30 seconds from its current position, whichever is less. This method will do nothing when called before the player is loaded. This method is only used for stations started with *startStation* or *startShoutcast.*

Example:
```
AppMobi.player.ffwd();  //search forward 30 seconds
```

4.9.14 show()

This method will show the native appMobi player UI.

Example:
```
AppMobi.player.show();
```

4.9.15 hide()

This method will hide the native appMobi player UI.

Example:
```
AppMobi.player.hide();
```

4.9.16 setPosition([portrait X], [portrait Y], [landscape X], [landscape Y]

On devices with a larger screen resolution such as the iPad, this command allows the developer to set the position of the native player on the screen.

Example:
```
AppMobi.player.setPosition(224,188,322,228);
```

4.9.17 setColors([backColor], [fillColor], [doneColor], [playColor])

This method allows developers to change the colors of the progress bar of the appMobi native player. Each color can be six or seven characters which hold the hex value of a color. Once this command is called, these settings will persist for the length of the session.

Color	Description	Defaults
backColor	The background of the progress bar	black
fillColor	The downloading progress of the progress bar	yellow
doneColor	The color the progress bar will turn once all downloading is complete	green
playColor	The color the play head indicating where in the progress bar is currently playing	red

Example:
```
AppMobi.player.setColors("000000", "#999999", "999999", "#ffffff");
```

4.10 playingtrack Object

The *playingtrack* object provides a series of properties that contain information about media currently played by the application. Please note that this object currently only works with stations hosted by appMobi and played with the *startStation* or *startShoutcast* commands. Not all properties may have values for all streams. An event is fired when the currently playing track changes.

4.10.1 artist

This property returns the artist of the currently playing track, if there is an artist available.

4.10.2 title

This property returns the title of the currently playing track.

4.10.3 album

This property returns an album the currently playing track might be found on, if that information is available.

4.10.4 imageurl

This property returns a URL of the album cover image for the currently playing track. In general the image returned is 160x160 pixels. If no image is available, the *imageurl* property will be an empty.

4.10.5 Code Sample for playingtrack

The “appMobi.player.track.change” event is thrown when the current player switches to a new track. That way, developers can know to recheck the A*ppMobi.playingtrack* properties for new track information.

Example:

```
document.addEventListener("appMobi.player.track.change",onTrackChanged,false);
function onTrackChanged(e)
{
        //The track has just changed
        alert(“The new track’s artist: “ + AppMobi.playingtrack.artist);
        alert(“The new track’s title: “ + AppMobi.playingtrack.title);
        alert(“The new track’s album name: “ + AppMobi.playingtrack.album);
        alert(“The new track’s image: “ + AppMobi.playingtrack.imageurl);
}
```

4.11 stats Object

The *stats* object provides access to Google analytics. In order for statistics to work properly, you must have specified your analytics account ID when creating this specific application on the appMobi website.

4.11.1 logEvent([eventString])

This function will log events to your account for application analytics. It is generally suggested that you log events with a structure similar to the following.

```
AppMobi.stats.logEvent("/[app name].[app section].[app subsection].[event]");
```

For example, AppMobi automatically logs certain usages for you. When you call playPodcast, the following event will be logged:

```
“/appMobi.podcast.[podcast name].start”
```

When the video podcast has finished playing the following will be logged:

```
"/appMobi.podcast.[podcast name].stop"
```

This kind of event lets you analyze how people are using your application.

4.12 Events

The appMobi library will throw the following events in response to certain device changes. All appMobi events follow a specific naming convention. They will always be *appMobi.SECTION.EVENT* (e.g. "*appMobi.device.ready*") or *appMobi.SECTION.SUBSECTION.EVENT* (e.g. "*appMobi.player.track.change*").

Please note that although the main functions are called from the global object *AppMobi*, events are referenced from a root of *appMobi* instead. Take note of the capitalization difference between the two since it can be confusing.

These events can be captured by creating event listeners for them in Javascript. For example:

```
document.addEventListener("[eventname]",on[Event],false);
function on[Event]()
{
  ...
}
```

4.12.1 appMobi.device.ready

This event will fire once all *AppMobi* library information is loaded. Look for this event to fire before attempting any *AppMobi* commands.

4.12.2 appMobi.device.orientation.change

This event will fire whenever the current orientation of the device changes. The value is sent with the event and updated in *AppMobi.device.orientation*. The *orientation* parameter on the event object will contain one of the following values:

Orientation	**Value**
Portrait	0 or 180
Landscape	90 or -90

4.12.3 appMobi.cache.media.add

This event fires once a file is added to the local file cache using the *AppMobi.cache.addToMediaCache* command.

4.12.4 appMobi.cache.media.remove

This event fires once a file is removed from the local file cache using the *AppMobi.cache.removeFromMediaCache* command.

4.12.5 appMobi.cache.media.clear

This event fires once a file is removed from the local file cache using the *AppMobi.cache.clearMediaCache* command.

4.12.6 appMobi.device.connection.update

This event is fired in response to the *AppMobi.device.updateConnection* command. The *connection* parameter on the event object will contain one of the following values:

Values	**Connection**
wifi	The device has an active wifi connection
cell	The device has an active cellular connection
none	The device does not currently have an Internet connection

4.12.7 appMobi.device.suspend

When an application is minimized, this event will be fired as soon as possible to alert the application that it is losing the user's focus. When the application reloads, be aware that you might see an *appMobi.device.resume* rather than the typical *appMobi.device.ready* event.

4.12.8 appMobi.device.resume

If an application was minimized, but it never left memory, this event will fire in lieu of the *appMobi.device.ready* command.

4.12.9 appMobi.device.hardware.back

If the *AppMobi.device.addVirtualPage* command has been called, and a virtual page is available to be trapped, pressing the hardware back button will fire this event rather than the default functionality of the back button. The hardware back button refers to the physical button on the device, and so it is obviously not available on all platforms (for example, the iPhone does not have a back button).

4.12.10 appMobi.device.remote.close

The *AppMobi.device.showRemoteSite* command will fire this event once a user has closed the new webview.

4.12.11 appMobi.notification.push.send

This event is fired once a push notification is sent from an application. It includes an event object having a "success" property that will hold either a true or a false. The event object may additionally include a "message" property describing any errors.

4.12.12 appMobi.notification.push.read

This event is fired once a push notification is marked as read and removed from the messaging system. It includes an event object having a "success" property that will hold either a true or a false. The event object may additionally include a “message” property describing any errors.

4.12.13 appMobi.notification.push.receive

This event is fired once the application has gotten a push notification. It includes an event object having a “success” property that will hold either a true or a false. The event object may additionally include a “message” property describing any errors.

4.12.14 appMobi.notification.push.refresh

This event is fired once the notification system refreshes the data on an application. It includes an event object having a “success” property that will hold either a true or a false. The event object may additionally include a “message” property describing any errors.

4.12.15 appMobi.notification.push.user.find

This event is fired following a request by an application to find other users of the application. It includes an event object having a “success” property that will hold either a true or a false. The event object may additionally include a “message” property describing any errors.

4.12.16 appMobi.notification.push.user.editattributes

This event is fired once further demographic information is associated with a user. It includes an event object having a “success” property that will hold either a true or a false. The event object may additionally include a “message” property describing any errors.

4.12.17 appMobi.notification.push.sendpassword

This event is fired once a request is made by a user to have their notifications password emailed to them. It includes an event object having a “success” property that will hold either a true or a false. The event object may additionally include a “message” property describing any errors.

4.12.18 appMobi.notification.push.user.edit

This event is fired once a user changes their email address or password in the notification system. It includes an event object having a “success” property that will hold either a true or a false. The event object may additionally include a “message” property describing any errors.

4.12.19 appMobi.notification.push.enable

This event is fired once a user’s credentials have been confirmed by the application and push notifications are enabled. It includes an event object having a “success” property that will hold either a true or a false. The event object may additionally include a “message” property describing any errors.

4.12.20 appMobi.player.sound.error

This event fires if a file referenced by the *AppMobi.player.playSound* command is invalid or missing.

4.12.21 appMobi.player.audio.stop

This event will fire once an audio file played using *AppMobi.player.startAudio* is complete or if *AppMobi.player.stopAudio* is called.

4.12.22 appMobi.player.audio.error

This event fires when there is an unknown error calling the *AppMobi.player.startAudio* command.

4.12.23 appMobi.player.audio.start

This event is fired when an audio file is started using the *AppMobi.player.startAudio* command.

4.12.24 appMobi.player.audio.busy

This event will fire if the *AppMobi.player.startAudio* command is called but the device is already playing a podcast or station or shoutcast station.

4.12.25 appMobi.player.podcast.stop

This event will fire once a video or audio podcast started with *AppMobi.player.playPodcast* is complete.

4.12.26 appMobi.player.podcast.error

This event is fired when the *AppMobi.player.playPodcast* command experiences an unknown error.

4.12.27 appMobi.player.podcast.start

This event is fired when a video is started using the *AppMobi.player.playPodcast* command.

4.12.28 appMobi.player.podcast.busy

This event will fire if the *AppMobi.player.playPodcast* command is called but the device is already playing a podcast or station or shoutcast station or an audio file.

4.12.29 appMobi.player.station.stop

This event is fired when a station stops because the user hit the stop button from the player screen or *AppMobi.player.stop* was called.

4.12.30 appMobi.player.station.error

This event is fired when the *AppMobi.player.startStation* command experiences an unknown error.

4.12.31 appMobi.player.station.start

This event is fired when a station is started.

4.12.32 appMobi.player.station.busy

This event will fire if the *AppMobi.player.startStation* command is called but the device is already playing a podcast or an audio file.

4.12.33 appMobi.player.station.pause

This event is fired when the *AppMobi.player.pause* command is called or when a station is paused from the player screen.

4.12.34 appMobi.player.station.play

This event is fired when the *AppMobi.player.play* command is called or when a station starts to play from the player screen.

4.12.35 appMobi.player.shoutcast.stop

This event is fired when a shoutcast station stops because the user hit the stop button from the player screen or *AppMobi.player.stop* was called.

4.12.36 appMobi.player.shoutcast.error

This event is fired when the *AppMobi.player.startShoutcast* command experiences an unknown error.

4.12.37 appMobi.player.shoutcast.start

This event is fired when a shoutcast station is started.

4.12.38 appMobi.player.shoutcast.busy

This event will fire if the *AppMobi.player.startShoutcast* command is called but the device is already playing a podcast or an audio file.

4.12.39 appMobi.player.shoutcast.pause

This event is fired when the *AppMobi.player.pause* command is called or when a shoutcast station is paused from the player screen.

4.12.40 appMobi.player.shoutcast.play

This event is fired when the *AppMobi.player.play* command is called or when a shoutcast station starts to play from the player screen.

4.12.41 appMobi.player.track.change

This event will fire each time a playing station changes to a new track and updates its metadata. This will happen for stations started using *AppMobi.player.startStation* or *AppMobi.player.startShoutcast.*

4.12.42 appMobi.player.hide

This event will fire when the appMobi player screen is hidden regardless of whether it was dismissed using the *AppMobi.player.hide* command or from the back button on the player itself.

4.12.43 appMobi.player.show

This event will fire when the appMobi player screen is shown. It can be triggered when appMobi itself shows the player such as after a command such as *AppMobi.player.startStation* is successfully executed or if a command such as *AppMobi.player.show* is run.

5.0 AUDIO PRIORITIES

With so many different methods of playing audio, there are circumstances where transitioning between two types of audio may have unexpected results. This matrix demonstrates the expected behavior when appMobi is playing one type of audio or video and then starts a second type of media.

Second Command

First Command

	Podcast	**Station**	**Shoutcast**	**Audio**	**Sound**
Podcast	B	B	B	B	Ok
Station	B	Sw	Sw	B	Ok
Shoutcast	B	Sw	Sw	B	Ok
Audio	B	B	B	Sw	Ok
Sound	Ok	Ok	Ok	Ok	Ok

Key:

Sw = Switch; The second command stops the first media and automatically switches to the second
B = Busy; The second command is not executed, the first media continues, and the *appMobi.player.audio.busy* event is fired
Ok = Okay; The sound command is neither interrupts if called second nor is able to be interrupted if called first.

6.0 CUSTOM IMAGES

The underlying application images that are provided by default may be changed or “skinned”. To do so, simply add your own versions of the following files into the _*appMobi* subdirectory of your application bundle.

Image	**Image Name**
player_bg_ls.png	Player background – landscape
player_bg_port.png	Player background – portrait
player_back_button.png	The button to return from the player
artwork_unavailable.png	The image shown when track artwork is not available
go_back.png	The button to go leave the player
go_live.png	The button to take the play head to the most current data
live.png	The prompt that shows on the right hand side of the player when the user is listening to the most current data
loading.png	The image shown as track artwork is loading
next_button.png	The button to go to the next track
pause_button.png	The button to pause playing
play_button.png	The button to restart playing
prev_button.png	The button to go to the previous track
retrieving_data.png	The image shown as track data is loading
snap_current_button.png	The button to take the user to the most recent data
splash_screen.png	The image to show as the application loads
stop_button.png	The button to stop playback
timer_button.png	The button to view the sleep timer
timer_button_on.png	The selected state of the sleep timer button
remote_close.png	The button to close a remote site

7.0 SUPPORTED FILE TYPES

This matrix shows all the supported media types for particular AppMobi platforms.

iPhone

Video Formats	H.264, MPEG-4 in .mp4, .m4v, .mov
Audio Formats	AAC, MP3, M4a

Android (Droid)

Video Formats	WMV, MPEG4, H.263, H.264.
Audio Formats	MP3, OGG