1. **src/com/example/myplugin/CalendarPlugin.java**

```
package com.example.myplugin;


import org.apache.cordova.api.CallbackContext;
import org.apache.cordova.api.CordovaPlugin;
import org.json.JSONObject;
import org.json.JSONArray;
import org.json.JSONException;
import android.app.Activity;
import android.content.Intent;


public class CalendarPlugin extends CordovaPlugin {
  public static final String ACTION_ADD_CALENDAR_ENTRY = "addCalendarEntry";


  @Override
  public boolean execute(String action, JSONArray args, CallbackContext callbackContext) throws JSONException {
    try {
      if (ACTION_ADD_CALENDAR_ENTRY.equals(action)) {
        JSONObject arg_object = args.getJSONObject(0);
        Intent calIntent = new Intent(Intent.ACTION_EDIT)
          .setType("vnd.android.cursor.item/event")
          .putExtra("beginTime", arg_object.getLong("startTimeMillis"))
          .putExtra("endTime", arg_object.getLong("endTimeMillis"))
          .putExtra("title", arg_object.getString("title"))
          .putExtra("description", arg_object.getString("description"))
          .putExtra("eventLocation", arg_object.getString("eventLocation"));

          this.cordova.getActivity().startActivity(calIntent);
          callbackContext.success();
          return true;
      }
      callbackContext.error("Invalid action");
      return false;
    } catch(Exception e) {
      System.err.println("Exception: " + e.getMessage());
      callbackContext.error(e.getMessage());
```

```
            return false;
        }
    }
}
```

2. **src/com/example/myplugin/CalendarPluginSampleApp.java**

```
package com.example.myplugin;


import android.os.Bundle;
import org.apache.cordova.*;


public class CalendarPluginSampleApp extends DroidGap
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // Set by <content src="index.html" /> in config.xml
        super.loadUrl(Config.getStartUrl());
        //super.loadUrl("file:///android_asset/www/index.html")
    }
}
```

3. **res/layout/main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Hello World, CalendarPluginSampleApp"
   />
</LinearLayout>
```

4. **res/values/strings.xml**

```
<?xml version='1.0' encoding='utf-8'?>
```

```
<resources>
  <string name="app_name">CalendarPluginSampleApp</string>
</resources>
```

5. **res/xml/config.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="io.cordova.helloCordova" version="2.0.0" xmlns="http://www.w3.org/ns/widgets">
  <name>Hello Cordova</name>
  <description>
    A sample Apache Cordova application that responds to the deviceready event.
  </description>
  <author email="dev@cordova.apache.org" href="http://cordova.io">
    Apache Cordova Team
  </author>
  <content src="index.html" />
  <feature name="App">
    <param name="android-package" value="org.apache.cordova.App" />
  </feature>
  <feature name="Geolocation">
    <param name="android-package" value="org.apache.cordova.GeoBroker" />
  </feature>
  <feature name="Device">
    <param name="android-package" value="org.apache.cordova.Device" />
  </feature>
  <feature name="Accelerometer">
    <param name="android-package"
value="org.apache.cordova.AccelListener" />
  </feature>
  <feature name="Compass">
    <param name="android-package"
value="org.apache.cordova.CompassListener" />
  </feature>
  <feature name="Media">
    <param name="android-package"
value="org.apache.cordova.AudioHandler" />
  </feature>
  <feature name="Camera">
    <param name="android-package"
value="org.apache.cordova.CameraLauncher" />
```

```
  </feature>
  <feature name="Contacts">
    <param name="android-package"
value="org.apache.cordova.ContactManager" />
  </feature>
  <feature name="File">
    <param name="android-package" value="org.apache.cordova.FileUtils" />
  </feature>
  <feature name="NetworkStatus">
    <param name="android-package"
value="org.apache.cordova.NetworkManager" />
  </feature>
  <feature name="Notification">
    <param name="android-package" value="org.apache.cordova.Notification"
/>
  </feature>
  <feature name="Storage">
    <param name="android-package" value="org.apache.cordova.Storage" />
  </feature>
  <feature name="FileTransfer">
```

```
        <param name="android-package" value="org.apache.cordova.FileTransfer"
/>
    </feature>
    <feature name="Capture">
        <param name="android-package" value="org.apache.cordova.Capture" />
    </feature>
    <feature name="Battery">
        <param name="android-package"
value="org.apache.cordova.BatteryListener" />
    </feature>
    <feature name="SplashScreen">
        <param name="android-package" value="org.apache.cordova.SplashScreen"
/>
    </feature>
    <feature name="Echo">
        <param name="android-package" value="org.apache.cordova.Echo" />
    </feature>
    <feature name="Globalization">
        <param name="android-package" value="org.apache.cordova.Globalization"
/>
    </feature>
```

```
  <feature name="InAppBrowser">
    <param name="android-package"
value="org.apache.cordova.InAppBrowser" />
  </feature>
  <feature name="CalendarPlugin">
    <param name="android-package"
value="com.example.myplugin.CalendarPlugin" />
  </feature>
  <plugins>
  </plugins>
  <access origin="http://127.0.0.1*" />
  <preference name="useBrowserHistory" value="true" />
  <preference name="exit-on-suspend" value="false" />
  <preference name="permissions" value="none" />
  <preference name="phonegap-version" value="2.9.0" />
  <preference name="orientation" value="default" />
  <preference name="target-device" value="universal" />
  <preference name="fullscreen" value="true" />
  <preference name="webviewbounce" value="true" />
  <preference name="prerendered-icon" value="true" />
```

```
    <preference name="stay-in-webview" value="false" />
    <preference name="ios-statusbarstyle" value="black-opaque" />
    <preference name="detect-data-types" value="true" />
    <preference name="show-splash-screen-spinner" value="true" />
    <preference name="auto-hide-splash-screen" value="true" />
    <preference name="disable-cursor" value="false" />
    <preference name="android-minSdkVersion" value="7" />
    <preference name="android-installLocation" value="auto" />
</widget>
```

6. AndroidManifest.xml

```
<?xml version='1.0' encoding='utf-8'?>
<manifest android:hardwareAccelerated="true" android:versionCode="1"
android:versionName="1.0.0" android:windowSoftInputMode="adjustPan"
package="com.example.myplugin"
xmlns:android="http://schemas.android.com/apk/res/android">
    <supports-screens android:anyDensity="true" android:largeScreens="true"
android:normalScreens="true" android:resizeable="true"
android:smallScreens="true" android:xlargeScreens="true" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMAN
DS" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.RECORD_VIDEO" />
    <uses-permission
android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS"
/>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
```

```
  <uses-permission
android:name="android.permission.BROADCAST_STICKY" />
  <application android:debuggable="true" android:hardwareAccelerated="true"
android:icon="@drawable/icon" android:label="@string/app_name">
    <activity
android:configChanges="orientation|keyboardHidden|keyboard|screenSize|locale"
android:label="@string/app_name" android:name="CalendarPluginSampleApp"
android:theme="@android:style/Theme.Black.NoTitleBar">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
  <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="17" />
</manifest>
```

7. **build.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="CalendarPluginSampleApp" default="help">
```

```
<!-- The local.properties file is created and updated by the 'android' tool.
     It contains the path to the SDK. It should *NOT* be checked into
     Version Control Systems. -->
<property file="local.properties" />

<!-- The ant.properties file can be created by you. It is only edited by the
     'android' tool to add properties to it.
     This is the place to change some Ant specific build properties.
     Here are some properties you may want to change/update:

     source.dir
         The name of the source directory. Default is 'src'.
     out.dir
         The name of the output directory. Default is 'bin'.

     For other overridable properties, look at the beginning of the rules
     files in the SDK, at tools/ant/build.xml

     Properties related to the SDK location or the project target should
```

```
    be updated using the 'android' tool with the 'update' action.

    This file is an integral part of the build system for your
    application and should be checked into Version Control Systems.

    -->
<property file="ant.properties" />

<!-- if sdk.dir was not set from one of the property file, then
    get it from the ANDROID_HOME env var.
    This must be done before we load project.properties since
    the proguard config can use sdk.dir -->
<property environment="env" />
<condition property="sdk.dir" value="${env.ANDROID_HOME}">
   <isset property="env.ANDROID_HOME" />
</condition>

<!-- The project.properties file is created and updated by the 'android'
    tool, as well as ADT.
```

```
        This contains project specific properties such as project target, and library

        dependencies. Lower level build properties are stored in ant.properties

        (or in .classpath for Eclipse projects).


        This file is an integral part of the build system for your

        application and should be checked into Version Control Systems. -->

    <loadproperties srcFile="project.properties" />


    <!-- quick check on sdk.dir -->

    <fail

            message="sdk.dir is missing. Make sure to generate local.properties using 'android update project' or to inject it through the ANDROID_HOME environment variable."

            unless="sdk.dir"

    />


    <!--

        Import per project custom build rules if present at the root of the project.

        This is the place to put custom intermediary targets such as:
            -pre-build
            -pre-compile
            -post-compile (This is typically used for code obfuscation.
                           Compiled code location: ${out.classes.absolute.dir}
                           If this is not done in place, override ${out.dex.input.absolute.dir})
            -post-package
            -post-build
            -pre-clean
    -->
    <import file="custom_rules.xml" optional="true" />

    <!-- Import the actual build file.

         To customize existing targets, there are two options:
         - Customize only one target:
             - copy/paste the target into this file, *before* the
               <import> task.
             - customize it to your needs.
         - Customize the whole content of build.xml
```

```
            - copy/paste the content of the rules files (minus the top node)
              into this file, replacing the <import> task.
            - customize to your needs.

        ***********************
        ****** IMPORTANT ******
        ***********************
        In all cases you must update the value of version-tag below to read 'custom'
instead of an integer,
        in order to avoid having your file be overridden by tools such as "android
update project"
    -->
    <!-- version-tag: 1 -->
    <import file="${sdk.dir}/tools/ant/build.xml" />

</project>
```

8. assets/www/config.xml

```
<?xml version='1.0' encoding='utf-8'?>
```

```
<widget id="com.example.myplugin" version="1.0.0"
xmlns="http://www.w3.org/ns/widgets"
xmlns:gap="http://phonegap.com/ns/1.0">
    <name>CalendarPluginSampleApp</name>
    <description>
        Hello World sample application that responds to the deviceready event.
    </description>
    <author email="support@phonegap.com" href="http://phonegap.com">
        PhoneGap Team
    </author>
    <feature name="http://api.phonegap.com/1.0/device" />
    <preference name="permissions" value="none" />
    <preference name="phonegap-version" value="2.9.0" />
    <preference name="orientation" value="default" />
    <preference name="target-device" value="universal" />
    <preference name="fullscreen" value="true" />
    <preference name="webviewbounce" value="true" />
    <preference name="prerendered-icon" value="true" />
    <preference name="stay-in-webview" value="false" />
    <preference name="ios-statusbarstyle" value="black-opaque" />
    <preference name="detect-data-types" value="true" />
    <preference name="exit-on-suspend" value="false" />
    <preference name="show-splash-screen-spinner" value="true" />
    <preference name="auto-hide-splash-screen" value="true" />
    <preference name="disable-cursor" value="false" />
    <preference name="android-minSdkVersion" value="7" />
    <preference name="android-installLocation" value="auto" />
    <icon src="icon.png" />
    <icon gap:density="ldpi" gap:platform="android" src="res/icon/android/icon-36-ldpi.png" />
    <icon gap:density="mdpi" gap:platform="android" src="res/icon/android/icon-48-mdpi.png" />
    <icon gap:density="hdpi" gap:platform="android" src="res/icon/android/icon-72-hdpi.png" />
    <icon gap:density="xhdpi" gap:platform="android" src="res/icon/android/icon-96-xhdpi.png" />
    <icon gap:platform="blackberry" src="res/icon/blackberry/icon-80.png" />
    <icon gap:platform="blackberry" gap:state="hover" src="res/icon/blackberry/icon-80.png" />
    <icon gap:platform="ios" height="57" src="res/icon/ios/icon-57.png" width="57" />
```

```
   <icon gap:platform="ios" height="72" src="res/icon/ios/icon-72.png" width="72" />

   <icon gap:platform="ios" height="114" src="res/icon/ios/icon-57-2x.png" width="114" />

   <icon gap:platform="ios" height="144" src="res/icon/ios/icon-72-2x.png" width="144" />

   <icon gap:platform="webos" src="res/icon/webos/icon-64.png" />

   <icon gap:platform="winphone" src="res/icon/windows-phone/icon-48.png" />

   <icon gap:platform="winphone" gap:role="background" src="res/icon/windows-phone/icon-173.png" />

   <gap:splash gap:density="ldpi" gap:platform="android" src="res/screen/android/screen-ldpi-portrait.png" />

   <gap:splash gap:density="mdpi" gap:platform="android" src="res/screen/android/screen-mdpi-portrait.png" />

   <gap:splash gap:density="hdpi" gap:platform="android" src="res/screen/android/screen-hdpi-portrait.png" />

   <gap:splash gap:density="xhdpi" gap:platform="android" src="res/screen/android/screen-xhdpi-portrait.png" />

   <gap:splash gap:platform="blackberry" src="res/screen/blackberry/screen-225.png" />

   <gap:splash gap:platform="ios" height="480" src="res/screen/ios/screen-iphone-portrait.png" width="320" />
```

```
    <gap:splash gap:platform="ios" height="960" src="res/screen/ios/screen-iphone-portrait-2x.png" width="640" />
    <gap:splash gap:platform="ios" height="1024" src="res/screen/ios/screen-ipad-portrait.png" width="768" />
    <gap:splash gap:platform="ios" height="768" src="res/screen/ios/screen-ipad-landscape.png" width="1024" />
    <gap:splash gap:platform="winphone" src="res/screen/windows-phone/screen-portrait.jpg" />
    <access origin="http://127.0.0.1*" />
</widget>
```

9. assets/www/cordova.js

```
// Platform: android
// 2.9.0-0-g83dc4bd
/*
 Licensed to the Apache Software Foundation (ASF) under one
 or more contributor license agreements.  See the NOTICE file
 distributed with this work for additional information
 regarding copyright ownership.  The ASF licenses this file
 to you under the Apache License, Version 2.0 (the
 "License"); you may not use this file except in compliance
 with the License.  You may obtain a copy of the License at

     http://www.apache.org/licenses/LICENSE-2.0

 Unless required by applicable law or agreed to in writing,
 software distributed under the License is distributed on an
 "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 KIND, either express or implied.  See the License for the
 specific language governing permissions and limitations
 under the License.
*/
;(function() {
var CORDOVA_JS_BUILD_LABEL = '2.9.0-0-g83dc4bd';
// file: lib/scripts/require.js

var require,
    define;

(function () {
    var modules = {},
    // Stack of moduleIds currently being built.
        requireStack = [],
    // Map of module ID -> index into requireStack of modules currently being built.
        inProgressModules = {},
        SEPERATOR = ".";


    function build(module) {
        var factory = module.factory,
            localRequire = function (id) {
                var resultantId = id;
                //Its a relative path, so lop off the last portion and add the id (minus "./")
                if (id.charAt(0) === ".") {
                    resultantId = module.id.slice(0, module.id.lastIndexOf(SEPERATOR)) + SEPERATOR + id.slice(2);
                }
                return require(resultantId);
            };
        module.exports = {};
        delete module.factory;
        factory(localRequire, module.exports, module);
        return module.exports;
    }

    require = function (id) {
        if (!modules[id]) {
            throw "module " + id + " not found";
        } else if (id in inProgressModules) {
            var cycle = requireStack.slice(inProgressModules[id]).join('->') + '->' + id;
            throw "Cycle in require graph: " + cycle;
        }
        if (modules[id].factory) {
            try {
                inProgressModules[id] = requireStack.length;
                requireStack.push(id);
                return build(modules[id]);
            } finally {
                delete inProgressModules[id];
                requireStack.pop();
            }
        }
        return modules[id].exports;
    };

    define = function (id, factory) {
        if (modules[id]) {
            throw "module " + id + " already defined";
        }

        modules[id] = {
            id: id,
            factory: factory
        };
    };
```

```
    define.remove = function (id) {
        delete modules[id];
    };

    define.moduleMap = modules;
})();

//Export for use in node
if (typeof module === "object" && typeof require === "function") {
    module.exports.require = require;
    module.exports.define = define;
}

// file: lib/cordova.js
define("cordova", function(require, exports, module) {


var channel = require('cordova/channel');
```

```
/**
 * Listen for DOMContentLoaded and notify our channel subscribers.
 */
document.addEventListener('DOMContentLoaded', function() {
   channel.onDOMContentLoaded.fire();
}, false);
if (document.readyState == 'complete' || document.readyState == 'interactive') {
   channel.onDOMContentLoaded.fire();
}

/**
 * Intercept calls to addEventListener + removeEventListener and handle deviceready,
 * resume, and pause events.
 */
var m_document_addEventListener = document.addEventListener;
var m_document_removeEventListener = document.removeEventListener;
var m_window_addEventListener = window.addEventListener;
var m_window_removeEventListener = window.removeEventListener;
```

```
/**
 * Houses custom event handlers to intercept on document + window event listeners.
 */
var documentEventHandlers = {},
    windowEventHandlers = {};

document.addEventListener = function(evt, handler, capture) {
    var e = evt.toLowerCase();
    if (typeof documentEventHandlers[e] != 'undefined') {
        documentEventHandlers[e].subscribe(handler);
    } else {
        m_document_addEventListener.call(document, evt, handler, capture);
    }
};

window.addEventListener = function(evt, handler, capture) {
    var e = evt.toLowerCase();
    if (typeof windowEventHandlers[e] != 'undefined') {
        windowEventHandlers[e].subscribe(handler);
    } else {
        m_window_addEventListener.call(window, evt, handler, capture);
    }
};

document.removeEventListener = function(evt, handler, capture) {
    var e = evt.toLowerCase();
    // If unsubscribing from an event that is handled by a plugin
    if (typeof documentEventHandlers[e] != "undefined") {
        documentEventHandlers[e].unsubscribe(handler);
    } else {
        m_document_removeEventListener.call(document, evt, handler, capture);
    }
};

window.removeEventListener = function(evt, handler, capture) {
    var e = evt.toLowerCase();
    // If unsubscribing from an event that is handled by a plugin
    if (typeof windowEventHandlers[e] != "undefined") {
        windowEventHandlers[e].unsubscribe(handler);
    } else {
        m_window_removeEventListener.call(window, evt, handler, capture);
    }
};


function createEvent(type, data) {
    var event = document.createEvent('Events');
    event.initEvent(type, false, false);
    if (data) {
        for (var i in data) {
            if (data.hasOwnProperty(i)) {
                event[i] = data[i];
            }
        }
    }
    return event;
}
```

```
if(typeof window.console === "undefined") {
    window.console = {
        log:function(){}
    };
}

var cordova = {
    define:define,
    require:require,
    /**
     * Methods to add/remove your own addEventListener hijacking on document + window.
     */
    addWindowEventHandler:function(event) {
        return (windowEventHandlers[event] = channel.create(event));
    },
    addStickyDocumentEventHandler:function(event) {
        return (documentEventHandlers[event] = channel.createSticky(event));
    },
    addDocumentEventHandler:function(event) {
        return (documentEventHandlers[event] = channel.create(event));
    },
    removeWindowEventHandler:function(event) {
        delete windowEventHandlers[event];
    },
    removeDocumentEventHandler:function(event) {
        delete documentEventHandlers[event];
    },
    /**
     * Retrieve original event handlers that were replaced by Cordova
     *
     * @return object
     */
    getOriginalHandlers: function() {
        return {'document': {'addEventListener': m_document_addEventListener, 'removeEventListener': m_document_removeEventListener},
        'window': {'addEventListener': m_window_addEventListener, 'removeEventListener': m_window_removeEventListener}};
    },
    /**
```

```
     * Method to fire event from native code
     * bNoDetach is required for events which cause an exception which needs to be caught in native code
     */
    fireDocumentEvent: function(type, data, bNoDetach) {
      var evt = createEvent(type, data);
      if (typeof documentEventHandlers[type] != 'undefined') {
        if( bNoDetach ) {
         documentEventHandlers[type].fire(evt);
        }
        else {
         setTimeout(function() {
           // Fire deviceready on listeners that were registered before cordova.js was loaded.
           if (type == 'deviceready') {
             document.dispatchEvent(evt);
           }
           documentEventHandlers[type].fire(evt);
         }, 0);
        }
        } else {
            document.dispatchEvent(evt);
        }
    },
    fireWindowEvent: function(type, data) {
        var evt = createEvent(type,data);
        if (typeof windowEventHandlers[type] != 'undefined') {
            setTimeout(function() {
                windowEventHandlers[type].fire(evt);
            }, 0);
        } else {
            window.dispatchEvent(evt);
        }
    },

    /**
     * Plugin callback mechanism.
     */
    // Randomize the starting callbackId to avoid collisions after refreshing or navigating.
    // This way, it's very unlikely that any new callback would get the same callbackId as an old callback.
    callbackId: Math.floor(Math.random() * 2000000000),
    callbacks:  {},
    callbackStatus: {
        NO_RESULT: 0,
        OK: 1,
        CLASS_NOT_FOUND_EXCEPTION: 2,
        ILLEGAL_ACCESS_EXCEPTION: 3,
        INSTANTIATION_EXCEPTION: 4,
        MALFORMED_URL_EXCEPTION: 5,
        IO_EXCEPTION: 6,
        INVALID_ACTION: 7,
        JSON_EXCEPTION: 8,
        ERROR: 9
    },

    /**
     * Called by native code when returning successful result from an action.
     */
    callbackSuccess: function(callbackId, args) {
        try {
            cordova.callbackFromNative(callbackId, true, args.status, [args.message], args.keepCallback);
        } catch (e) {
            console.log("Error in error callback: " + callbackId + " = "+e);
        }
    },

    /**
     * Called by native code when returning error result from an action.
     */
    callbackError: function(callbackId, args) {
        // TODO: Deprecate callbackSuccess and callbackError in favour of callbackFromNative.
        // Derive success from status.
        try {
            cordova.callbackFromNative(callbackId, false, args.status, [args.message], args.keepCallback);
        } catch (e) {
```

```
      console.log("Error in error callback: " + callbackId + " = "+e);
    }
  },

  /**
   * Called by native code when returning the result from an action.
   */
  callbackFromNative: function(callbackId, success, status, args, keepCallback) {
    var callback = cordova.callbacks[callbackId];
    if (callback) {
      if (success && status == cordova.callbackStatus.OK) {
        callback.success && callback.success.apply(null, args);
      } else if (!success) {
        callback.fail && callback.fail.apply(null, args);
      }

      // Clear callback if not expecting any more results
      if (!keepCallback) {
        delete cordova.callbacks[callbackId];
```

```
            }
        }
    },
    addConstructor: function(func) {
        channel.onCordovaReady.subscribe(function() {
            try {
                func();
            } catch(e) {
                console.log("Failed to run constructor: " + e);
            }
        });
    }
};

// Register pause, resume and deviceready channels as events on document.
channel.onPause = cordova.addDocumentEventHandler('pause');
channel.onResume = cordova.addDocumentEventHandler('resume');
channel.onDeviceReady                                                    =
cordova.addStickyDocumentEventHandler('deviceready');
```

```
module.exports = cordova;

});

// file: lib/common/argscheck.js
define("cordova/argscheck", function(require, exports, module) {

var exec = require('cordova/exec');
var utils = require('cordova/utils');

var moduleExports = module.exports;

var typeMap = {
    'A': 'Array',
    'D': 'Date',
    'N': 'Number',
    'S': 'String',
    'F': 'Function',
    'O': 'Object'
```

```
};

function extractParamName(callee, argIndex) {
 return (/.*?\((.*?)\)/).exec(callee)[1].split(', ')[argIndex];
}

function checkArgs(spec, functionName, args, opt_callee) {
  if (!moduleExports.enableChecks) {
    return;
  }
  var errMsg = null;
  var typeName;
  for (var i = 0; i < spec.length; ++i) {
    var c = spec.charAt(i),
      cUpper = c.toUpperCase(),
      arg = args[i];
    // Asterix means allow anything.
    if (c == '*') {
      continue;
    }
    typeName = utils.typeName(arg);
    if ((arg === null || arg === undefined) && c == cUpper) {
      continue;
    }
    if (typeName != typeMap[cUpper]) {
      errMsg = 'Expected ' + typeMap[cUpper];
      break;
    }
  }
  if (errMsg) {
    errMsg += ', but got ' + typeName + '.';
    errMsg = 'Wrong type for parameter "' + extractParamName(opt_callee || args.callee, i) + '" of ' + functionName + ': ' + errMsg;
    // Don't log when running jake test.
    if (typeof jasmine == 'undefined') {
      console.error(errMsg);
    }
    throw TypeError(errMsg);
  }
```

```
}


function getValue(value, defaultValue) {

    return value === undefined ? defaultValue : value;

}


moduleExports.checkArgs = checkArgs;

moduleExports.getValue = getValue;

moduleExports.enableChecks = true;



});


// file: lib/common/builder.js

define("cordova/builder", function(require, exports, module) {


var utils = require('cordova/utils');


function each(objects, func, context) {
    for (var prop in objects) {
        if (objects.hasOwnProperty(prop)) {
            func.apply(context, [objects[prop], prop]);
        }
    }
}

function clobber(obj, key, value) {
    exports.replaceHookForTesting(obj, key);
    obj[key] = value;
    // Getters can only be overridden by getters.
    if (obj[key] !== value) {
        utils.defineGetter(obj, key, function() {
            return value;
        });
    }
}

function assignOrWrapInDeprecateGetter(obj, key, value, message) {
    if (message) {

        utils.defineGetter(obj, key, function() {

            console.log(message);

            delete obj[key];

            clobber(obj, key, value);

            return value;

        });

    } else {

        clobber(obj, key, value);

    }

}



function include(parent, objects, clobber, merge) {

    each(objects, function (obj, key) {

        try {

          var result = obj.path ? require(obj.path) : {};



          if (clobber) {

              // Clobber if it doesn't exist.
```

```
      if (typeof parent[key] === 'undefined') {
        assignOrWrapInDeprecateGetter(parent, key, result, obj.deprecated);
      } else if (typeof obj.path !== 'undefined') {
        // If merging, merge properties onto parent, otherwise, clobber.
        if (merge) {
          recursiveMerge(parent[key], result);
        } else {
          assignOrWrapInDeprecateGetter(parent, key, result, obj.deprecated);
        }
      }
      result = parent[key];
    } else {
     // Overwrite if not currently defined.
     if (typeof parent[key] == 'undefined') {
      assignOrWrapInDeprecateGetter(parent, key, result, obj.deprecated);
     } else {
      // Set result to what already exists, so we can build children into it if they exist.
      result = parent[key];
```

```
            }
          }

          if (obj.children) {
            include(result, obj.children, clobber, merge);
          }
        } catch(e) {
          utils.alert('Exception building cordova JS globals: ' + e + ' for key "' + key + '"');
        }
    });
}

/**
 * Merge properties from one object onto another recursively.  Properties from
 * the src object will overwrite existing target property.
 *
 * @param target Object to merge properties into.
 * @param src Object to merge properties from.
 */
```

```
function recursiveMerge(target, src) {
   for (var prop in src) {
      if (src.hasOwnProperty(prop)) {
         if (target.prototype && target.prototype.constructor === target) {
            // If the target object is a constructor override off prototype.
            clobber(target.prototype, prop, src[prop]);
         } else {
            if (typeof src[prop] === 'object' && typeof target[prop] === 'object') {
               recursiveMerge(target[prop], src[prop]);
            } else {
               clobber(target, prop, src[prop]);
            }
         }
      }
   }
}

exports.buildIntoButDoNotClobber = function(objects, target) {
   include(target, objects, false, false);
};

exports.buildIntoAndClobber = function(objects, target) {

    include(target, objects, true, false);

};

exports.buildIntoAndMerge = function(objects, target) {

    include(target, objects, true, true);

};

exports.recursiveMerge = recursiveMerge;

exports.assignOrWrapInDeprecateGetter = assignOrWrapInDeprecateGetter;

exports.replaceHookForTesting = function() {};


});


// file: lib/common/channel.js

define("cordova/channel", function(require, exports, module) {


var utils = require('cordova/utils'),

    nextGuid = 1;
```

```
/**
 * Custom pub-sub "channel" that can have functions subscribed to it
 * This object is used to define and control firing of events for
 * cordova initialization, as well as for custom events thereafter.
 *
 * The order of events during page load and Cordova startup is as follows:
 *
 * onDOMContentLoaded*         Internal event that is received when the web page is loaded and parsed.
 * onNativeReady*              Internal event that indicates the Cordova native side is ready.
 * onCordovaReady*             Internal event fired when all Cordova JavaScript objects have been created.
 * onCordovaInfoReady*         Internal event fired when device properties are available.
 * onCordovaConnectionReady*   Internal event fired when the connection property has been set.
 * onDeviceReady*              User event fired to indicate that Cordova is ready
 * onResume                    User event fired to indicate a start/resume lifecycle event
 * onPause                     User event fired to indicate a pause lifecycle event
 * onDestroy*                  Internal event fired when app is being destroyed (User should use window.onunload event, not this one).
 *
 * The events marked with an * are sticky. Once they have fired, they will stay in the fired state.
 * All listeners that subscribe after the event is fired will be executed right away.
 *
 * The only Cordova events that user code should register for are:
 *      deviceready           Cordova native code is initialized and Cordova APIs can be called from JavaScript
 *      pause                 App has moved to background
 *      resume                App has returned to foreground
 *
 * Listeners can be registered as:
 *      document.addEventListener("deviceready", myDeviceReadyListener, false);
 *      document.addEventListener("resume", myResumeListener, false);
 *      document.addEventListener("pause", myPauseListener, false);
 *
 * The DOM lifecycle events should be used for saving and restoring state
 *      window.onload
 *      window.onunload
 *
 */

/**
 * Channel
 * @constructor
 * @param type  String the channel name
 */
var Channel = function(type, sticky) {
    this.type = type;
    // Map of guid -> function.
    this.handlers = {};
    // 0 = Non-sticky, 1 = Sticky non-fired, 2 = Sticky fired.
    this.state = sticky ? 1 : 0;
    // Used in sticky mode to remember args passed to fire().
    this.fireArgs = null;
    // Used by onHasSubscribersChange to know if there are any listeners.
    this.numHandlers = 0;
```

```
        // Function that is called when the first listener is subscribed, or when
        // the last listener is unsubscribed.
        this.onHasSubscribersChange = null;
    },
        channel = {
            /**
             * Calls the provided function only after all of the channels specified
             * have been fired. All channels must be sticky channels.
             */
            join: function(h, c) {
                var len = c.length,
                    i = len,
                    f = function() {
                        if (!(--i)) h();
                    };
                for (var j=0; j<len; j++) {
                    if (c[j].state === 0) {
                        throw Error('Can only use join with sticky channels.');
                    }
                    c[j].subscribe(f);
                }
                if (!len) h();
            },
            create: function(type) {
                return channel[type] = new Channel(type, false);
            },
            createSticky: function(type) {
                return channel[type] = new Channel(type, true);
            },

            /**
             * cordova Channels that must fire before "deviceready" is fired.
             */
            deviceReadyChannelsArray: [],
            deviceReadyChannelsMap: {},

            /**
             * Indicate that a feature needs to be initialized before it is ready to be used.
```

```
     * This holds up Cordova's "deviceready" event until the feature has been initialized
     * and Cordova.initComplete(feature) is called.
     *
     * @param feature {String}     The unique feature name
     */
    waitForInitialization: function(feature) {
        if (feature) {
            var c = channel[feature] || this.createSticky(feature);
            this.deviceReadyChannelsMap[feature] = c;
            this.deviceReadyChannelsArray.push(c);
        }
    },

    /**
     * Indicate that initialization code has completed and the feature is ready to be used.
     *
     * @param feature {String}     The unique feature name
     */
    initializationComplete: function(feature) {
      var c = this.deviceReadyChannelsMap[feature];
      if (c) {
        c.fire();
      }
    }
  };

function forceFunction(f) {
  if (typeof f != 'function') throw "Function required as first argument!";
}

/**
 * Subscribes the given function to the channel. Any time that
 * Channel.fire is called so too will the function.
 * Optionally specify an execution context for the function
 * and a guid that can be used to stop subscribing to the channel.
 * Returns the guid.
 */
```

```
Channel.prototype.subscribe = function(f, c) {
    // need a function to call
    forceFunction(f);
    if (this.state == 2) {
        f.apply(c || this, this.fireArgs);
        return;
    }

    var func = f,
        guid = f.observer_guid;
    if (typeof c == "object") { func = utils.close(c, f); }

    if (!guid) {
        // first time any channel has seen this subscriber
        guid = '' + nextGuid++;
    }
    func.observer_guid = guid;
    f.observer_guid = guid;
    // Don't add the same handler more than once.
    if (!this.handlers[guid]) {
        this.handlers[guid] = func;
        this.numHandlers++;
        if (this.numHandlers == 1) {
            this.onHasSubscribersChange && this.onHasSubscribersChange();
        }
    }
};

/**
 * Unsubscribes the function with the given guid from the channel.
 */
Channel.prototype.unsubscribe = function(f) {
    // need a function to unsubscribe
    forceFunction(f);

    var guid = f.observer_guid,
        handler = this.handlers[guid];
    if (handler) {
        delete this.handlers[guid];
        this.numHandlers--;
        if (this.numHandlers === 0) {
            this.onHasSubscribersChange && this.onHasSubscribersChange();
        }
    }
};

/**
 * Calls all functions subscribed to this channel.
 */
Channel.prototype.fire = function(e) {
    var fail = false,
        fireArgs = Array.prototype.slice.call(arguments);
    // Apply stickiness.
    if (this.state == 1) {
        this.state = 2;
        this.fireArgs = fireArgs;
```

```
  }
  if (this.numHandlers) {
    // Copy the values first so that it is safe to modify it from within
    // callbacks.
    var toCall = [];
    for (var item in this.handlers) {
      toCall.push(this.handlers[item]);
    }
    for (var i = 0; i < toCall.length; ++i) {
      toCall[i].apply(this, fireArgs);
    }
    if (this.state == 2 && this.numHandlers) {
      this.numHandlers = 0;
      this.handlers = {};
      this.onHasSubscribersChange && this.onHasSubscribersChange();
    }
  }
};
```

```
// defining them here so they are ready super fast!
// DOM event that is received when the web page is loaded and parsed.
channel.createSticky('onDOMContentLoaded');

// Event to indicate the Cordova native side is ready.
channel.createSticky('onNativeReady');

// Event to indicate that all Cordova JavaScript objects have been created
// and it's time to run plugin constructors.
channel.createSticky('onCordovaReady');

// Event to indicate that device properties are available
channel.createSticky('onCordovaInfoReady');

// Event to indicate that the connection property has been set.
channel.createSticky('onCordovaConnectionReady');

// Event to indicate that all automatically loaded JS plugins are loaded and ready.
```

```
channel.createSticky('onPluginsReady');

// Event to indicate that Cordova is ready
channel.createSticky('onDeviceReady');

// Event to indicate a resume lifecycle event
channel.create('onResume');

// Event to indicate a pause lifecycle event
channel.create('onPause');

// Event to indicate a destroy lifecycle event
channel.createSticky('onDestroy');

// Channels that must fire before "deviceready" is fired.
channel.waitForInitialization('onCordovaReady');
channel.waitForInitialization('onCordovaConnectionReady');
channel.waitForInitialization('onDOMContentLoaded');
```

```
module.exports = channel;

});

// file: lib/common/commandProxy.js
define("cordova/commandProxy", function(require, exports, module) {


// internal map of proxy function
var CommandProxyMap = {};

module.exports = {

   // example: cordova.commandProxy.add("Accelerometer",{getCurrentAcceleration: function(successCallback, errorCallback, options) {...},...);
   add:function(id,proxyObj) {
       console.log("adding proxy for " + id);
       CommandProxyMap[id] = proxyObj;
       return proxyObj;
```

```
    },

    // cordova.commandProxy.remove("Accelerometer");
    remove:function(id) {
        var proxy = CommandProxyMap[id];
        delete CommandProxyMap[id];
        CommandProxyMap[id] = null;
        return proxy;
    },

    get:function(service,action) {
        return ( CommandProxyMap[service] ?
CommandProxyMap[service][action] : null );
    }
};
});

// file: lib/android/exec.js
define("cordova/exec", function(require, exports, module) {
```

```
/**
 * Execute a cordova command.  It is up to the native side whether this action
 * is synchronous or asynchronous.  The native side can return:
 *      Synchronous: PluginResult object as a JSON string
 *      Asynchronous: Empty string ""
 * If async, the native side will cordova.callbackSuccess or cordova.callbackError,
 * depending upon the result of the action.
 *
 * @param {Function} success    The success callback
 * @param {Function} fail       The fail callback
 * @param {String} service      The name of the service to use
 * @param {String} action       Action to be run in cordova
 * @param {String[]} [args]     Zero or more arguments to pass to the method
 */
var cordova = require('cordova'),
    nativeApiProvider = require('cordova/plugin/android/nativeapiprovider'),
    utils = require('cordova/utils'),
    jsToNativeModes = {
        PROMPT: 0,
        JS_OBJECT: 1,
        // This mode is currently for benchmarking purposes only. It must be enabled
        // on the native side through the ENABLE_LOCATION_CHANGE_EXEC_MODE
        // constant within CordovaWebViewClient.java before it will work.
        LOCATION_CHANGE: 2
    },
    nativeToJsModes = {
        // Polls for messages using the JS->Native bridge.
        POLLING: 0,
        // For LOAD_URL to be viable, it would need to have a work-around for
        // the bug where the soft-keyboard gets dismissed when a message is sent.
        LOAD_URL: 1,
        // For the ONLINE_EVENT to be viable, it would need to intercept all event
        // listeners (both through addEventListener and window.ononline) as well
        // as set the navigator property itself.
        ONLINE_EVENT: 2,
        // Uses reflection to access private APIs of the WebView that can send JS
        // to be executed.
        // Requires Android 3.2.4 or above.
```

```
        PRIVATE_API: 3
    },
    jsToNativeBridgeMode,  // Set lazily.
    nativeToJsBridgeMode = nativeToJsModes.ONLINE_EVENT,
    pollEnabled = false,
    messagesFromNative = [];

function androidExec(success, fail, service, action, args) {
    // Set default bridge modes if they have not already been set.
    // By default, we use the failsafe, since addJavascriptInterface breaks too often
    if (jsToNativeBridgeMode === undefined) {
        androidExec.setJsToNativeBridgeMode(jsToNativeModes.JS_OBJECT);
    }

    // Process any ArrayBuffers in the args into a string.
    for (var i = 0; i < args.length; i++) {
        if (utils.typeName(args[i]) == 'ArrayBuffer') {
            args[i] = window.btoa(String.fromCharCode.apply(null, new
Uint8Array(args[i])));
        }
```

```
}

var callbackId = service + cordova.callbackId++,
    argsJson = JSON.stringify(args);

if (success || fail) {
    cordova.callbacks[callbackId] = {success:success, fail:fail};
}

if (jsToNativeBridgeMode == jsToNativeModes.LOCATION_CHANGE) {
    window.location = 'http://cdv_exec/' + service + '#' + action + '#' + callbackId + '#' + argsJson;
} else {
    var messages = nativeApiProvider.get().exec(service, action, callbackId, argsJson);
    // If argsJson was received by Java as null, try again with the PROMPT bridge mode.
    // This happens in rare circumstances, such as when certain Unicode characters are passed over the bridge on a Galaxy S2.  See CB-2666.
    if (jsToNativeBridgeMode == jsToNativeModes.JS_OBJECT && messages === "@Null arguments.") {
            androidExec.setJsToNativeBridgeMode(jsToNativeModes.PROMPT);
            androidExec(success, fail, service, action, args);
            androidExec.setJsToNativeBridgeMode(jsToNativeModes.JS_OBJECT);
            return;
        } else {
            androidExec.processMessages(messages);
        }
    }
}

function pollOnce() {
    var msg = nativeApiProvider.get().retrieveJsMessages();
    androidExec.processMessages(msg);
}

function pollingTimerFunc() {
    if (pollEnabled) {
        pollOnce();
        setTimeout(pollingTimerFunc, 50);
    }
}

function hookOnlineApis() {
    function proxyEvent(e) {
        cordova.fireWindowEvent(e.type);
    }
    // The network module takes care of firing online and offline events.
    // It currently fires them only on document though, so we bridge them
    // to window here (while first listening for exec()-releated online/offline
    // events).
    window.addEventListener('online', pollOnce, false);
    window.addEventListener('offline', pollOnce, false);
    cordova.addWindowEventHandler('online');
    cordova.addWindowEventHandler('offline');
    document.addEventListener('online', proxyEvent, false);
    document.addEventListener('offline', proxyEvent, false);
}
```

```
hookOnlineApis();

androidExec.jsToNativeModes = jsToNativeModes;
androidExec.nativeToJsModes = nativeToJsModes;

androidExec.setJsToNativeBridgeMode = function(mode) {
  if (mode == jsToNativeModes.JS_OBJECT && !window._cordovaNative) {
    console.log('Falling back on PROMPT mode since _cordovaNative is missing. Expected for Android 3.2 and lower only.');
    mode = jsToNativeModes.PROMPT;
  }
  nativeApiProvider.setPreferPrompt(mode == jsToNativeModes.PROMPT);
  jsToNativeBridgeMode = mode;
};

androidExec.setNativeToJsBridgeMode = function(mode) {
  if (mode == nativeToJsBridgeMode) {
    return;
  }
  if (nativeToJsBridgeMode == nativeToJsModes.POLLING) {
        pollEnabled = false;
    }

    nativeToJsBridgeMode = mode;
    // Tell the native side to switch modes.
    nativeApiProvider.get().setNativeToJsBridgeMode(mode);

    if (mode == nativeToJsModes.POLLING) {
        pollEnabled = true;
        setTimeout(pollingTimerFunc, 1);
    }
};

// Processes a single message, as encoded by NativeToJsMessageQueue.java.
function processMessage(message) {
    try {
        var firstChar = message.charAt(0);
        if (firstChar == 'J') {
            eval(message.slice(1));
```

```
} else if (firstChar == 'S' || firstChar == 'F') {
    var success = firstChar == 'S';
    var keepCallback = message.charAt(1) == '1';
    var spaceIdx = message.indexOf(' ', 2);
    var status = +message.slice(2, spaceIdx);
    var nextSpaceIdx = message.indexOf(' ', spaceIdx + 1);
    var callbackId = message.slice(spaceIdx + 1, nextSpaceIdx);
    var payloadKind = message.charAt(nextSpaceIdx + 1);
    var payload;
    if (payloadKind == 's') {
        payload = message.slice(nextSpaceIdx + 2);
    } else if (payloadKind == 't') {
        payload = true;
    } else if (payloadKind == 'f') {
        payload = false;
    } else if (payloadKind == 'N') {
        payload = null;
    } else if (payloadKind == 'n') {
        payload = +message.slice(nextSpaceIdx + 2);
```

```
      } else if (payloadKind == 'A') {
        var data = message.slice(nextSpaceIdx + 2);
        var bytes = window.atob(data);
        var arraybuffer = new Uint8Array(bytes.length);
        for (var i = 0; i < bytes.length; i++) {
          arraybuffer[i] = bytes.charCodeAt(i);
        }
        payload = arraybuffer.buffer;
      } else if (payloadKind == 'S') {
        payload = window.atob(message.slice(nextSpaceIdx + 2));
      } else {
        payload = JSON.parse(message.slice(nextSpaceIdx + 1));
      }
      cordova.callbackFromNative(callbackId, success, status, [payload], keepCallback);
    } else {
      console.log("processMessage failed: invalid message:" + message);
    }
  } catch (e) {
    console.log("processMessage failed: Message: " + message);
```

```
        console.log("processMessage failed: Error: " + e);
        console.log("processMessage failed: Stack: " + e.stack);
    }
}

// This is called from the NativeToJsMessageQueue.java.
androidExec.processMessages = function(messages) {
    if (messages) {
        messagesFromNative.push(messages);
        // Check for the reentrant case, and enqueue the message if that's the case.
        if (messagesFromNative.length > 1) {
            return;
        }
        while (messagesFromNative.length) {
            // Don't unshift until the end so that reentrancy can be detected.
            messages = messagesFromNative[0];
            // The Java side can send a * message to indicate that it
            // still has messages waiting to be retrieved.
            if (messages == '*') {
                messagesFromNative.shift();
                window.setTimeout(pollOnce, 0);
                return;
            }

            var spaceIdx = messages.indexOf(' ');
            var msgLen = +messages.slice(0, spaceIdx);
            var message = messages.substr(spaceIdx + 1, msgLen);
            messages = messages.slice(spaceIdx + msgLen + 1);
            processMessage(message);
            if (messages) {
                messagesFromNative[0] = messages;
            } else {
                messagesFromNative.shift();
            }
        }
    }
};
```

```
module.exports = androidExec;

});

// file: lib/common/modulemapper.js
define("cordova/modulemapper", function(require, exports, module) {

var builder = require('cordova/builder'),
    moduleMap = define.moduleMap,
    symbolList,
    deprecationMap;

exports.reset = function() {
    symbolList = [];
    deprecationMap = {};
};

function addEntry(strategy, moduleName, symbolPath, opt_deprecationMessage) {
    if (!(moduleName in moduleMap)) {
        throw new Error('Module ' + moduleName + ' does not exist.');
    }
    symbolList.push(strategy, moduleName, symbolPath);
    if (opt_deprecationMessage) {
        deprecationMap[symbolPath] = opt_deprecationMessage;
    }
}

// Note: Android 2.3 does have Function.bind().
exports.clobbers = function(moduleName, symbolPath, opt_deprecationMessage)
{
    addEntry('c', moduleName, symbolPath, opt_deprecationMessage);
};

exports.merges = function(moduleName, symbolPath, opt_deprecationMessage) {
    addEntry('m', moduleName, symbolPath, opt_deprecationMessage);
};

exports.defaults = function(moduleName, symbolPath, opt_deprecationMessage)
{
    addEntry('d', moduleName, symbolPath, opt_deprecationMessage);
};

function prepareNamespace(symbolPath, context) {
    if (!symbolPath) {
        return context;
    }
    var parts = symbolPath.split('.');
    var cur = context;
    for (var i = 0, part; part = parts[i]; ++i) {
        cur = cur[part] = cur[part] || {};
    }
    return cur;
}

exports.mapModules = function(context) {
    var origSymbols = {};
    context.CDV_origSymbols = origSymbols;
    for (var i = 0, len = symbolList.length; i < len; i += 3) {
```

```
var strategy = symbolList[i];
var moduleName = symbolList[i + 1];
var symbolPath = symbolList[i + 2];
var lastDot = symbolPath.lastIndexOf('.');
var namespace = symbolPath.substr(0, lastDot);
var lastName = symbolPath.substr(lastDot + 1);

var module = require(moduleName);
var deprecationMsg = symbolPath in deprecationMap ? 'Access made to deprecated symbol: ' + symbolPath + '. ' + deprecationMsg : null;
var parentObj = prepareNamespace(namespace, context);
var target = parentObj[lastName];

if (strategy == 'm' && target) {
  builder.recursiveMerge(target, module);
} else if ((strategy == 'd' && !target) || (strategy != 'd')) {
  if (!(symbolPath in origSymbols)) {
    origSymbols[symbolPath] = target;
  }
```

```
            builder.assignOrWrapInDeprecateGetter(parentObj, lastName, module, deprecationMsg);
        }
    }
};

exports.getOriginalSymbol = function(context, symbolPath) {
    var origSymbols = context.CDV_origSymbols;
    if (origSymbols && (symbolPath in origSymbols)) {
        return origSymbols[symbolPath];
    }
    var parts = symbolPath.split('.');
    var obj = context;
    for (var i = 0; i < parts.length; ++i) {
        obj = obj && obj[parts[i]];
    }
    return obj;
};

exports.loadMatchingModules = function(matchingRegExp) {
    for (var k in moduleMap) {

        if (matchingRegExp.exec(k)) {

            require(k);

        }

    }

};


exports.reset();




});


// file: lib/android/platform.js

define("cordova/platform", function(require, exports, module) {


module.exports = {

    id: "android",

    initialize:function() {

        var channel = require("cordova/channel"),
            cordova = require('cordova'),
            exec = require('cordova/exec'),
            modulemapper = require('cordova/modulemapper');

        modulemapper.loadMatchingModules(/cordova.*\/symbols$/);
        modulemapper.clobbers('cordova/plugin/android/app', 'navigator.app');

        modulemapper.mapModules(window);

        // Inject a listener for the backbutton on the document.
        var backButtonChannel = cordova.addDocumentEventHandler('backbutton');
        backButtonChannel.onHasSubscribersChange = function() {
            // If we just attached the first handler or detached the last handler,
            // let native know we need to override the back button.
            exec(null, null, "App", "overrideBackbutton", [this.numHandlers == 1]);
        };

        // Add hardware MENU and SEARCH button handlers
        cordova.addDocumentEventHandler('menubutton');
```

```
    cordova.addDocumentEventHandler('searchbutton');

    // Let native code know we are all done on the JS side.
    // Native code will then un-hide the WebView.
    channel.join(function() {
      exec(null, null, "App", "show", []);
    }, [channel.onCordovaReady]);
  }
};

});

// file: lib/common/plugin/Acceleration.js
define("cordova/plugin/Acceleration", function(require, exports, module) {

var Acceleration = function(x, y, z, timestamp) {
  this.x = x;
  this.y = y;
  this.z = z;
```

```
    this.timestamp = timestamp || (new Date()).getTime();
};


module.exports = Acceleration;


});


// file: lib/common/plugin/Camera.js
define("cordova/plugin/Camera", function(require, exports, module) {


var argscheck = require('cordova/argscheck'),
    exec = require('cordova/exec'),
    Camera = require('cordova/plugin/CameraConstants'),
    CameraPopoverHandle = require('cordova/plugin/CameraPopoverHandle');


var cameraExport = {};


// Tack on the Camera Constants to the base camera plugin.
for (var key in Camera) {
   cameraExport[key] = Camera[key];
}

/**
 * Gets a picture from source defined by "options.sourceType", and returns the
 * image as defined by the "options.destinationType" option.

 * The defaults are sourceType=CAMERA and destinationType=FILE_URI.
 *
 * @param {Function} successCallback
 * @param {Function} errorCallback
 * @param {Object} options
 */
cameraExport.getPicture = function(successCallback, errorCallback, options) {
   argscheck.checkArgs('fFO', 'Camera.getPicture', arguments);
   options = options || {};
   var getValue = argscheck.getValue;

   var quality = getValue(options.quality, 50);
   var destinationType = getValue(options.destinationType,
Camera.DestinationType.FILE_URI);

   var sourceType = getValue(options.sourceType,
Camera.PictureSourceType.CAMERA);

   var targetWidth = getValue(options.targetWidth, -1);

   var targetHeight = getValue(options.targetHeight, -1);

   var encodingType = getValue(options.encodingType,
Camera.EncodingType.JPEG);

   var mediaType = getValue(options.mediaType,
Camera.MediaType.PICTURE);

   var allowEdit = !!options.allowEdit;

   var correctOrientation = !!options.correctOrientation;

   var saveToPhotoAlbum = !!options.saveToPhotoAlbum;

   var popoverOptions = getValue(options.popoverOptions, null);

   var cameraDirection = getValue(options.cameraDirection,
Camera.Direction.BACK);



   var args = [quality, destinationType, sourceType, targetWidth, targetHeight,
encodingType,

            mediaType, allowEdit, correctOrientation, saveToPhotoAlbum,
popoverOptions, cameraDirection];
```

```
    exec(successCallback, errorCallback, "Camera", "takePicture", args);
    return new CameraPopoverHandle();
};

cameraExport.cleanup = function(successCallback, errorCallback) {
    exec(successCallback, errorCallback, "Camera", "cleanup", []);
};

module.exports = cameraExport;

});

// file: lib/common/plugin/CameraConstants.js
define("cordova/plugin/CameraConstants", function(require, exports, module) {

module.exports = {
  DestinationType:{
    DATA_URL: 0,         // Return base64 encoded string
    FILE_URI: 1,         // Return file uri (content://media/external/images/media/2
for Android)
```

```
    NATIVE_URI: 2       // Return native uri (eg. asset-library://... for iOS)
  },
  EncodingType:{
    JPEG: 0,            // Return JPEG encoded image
    PNG: 1              // Return PNG encoded image
  },
  MediaType:{
    PICTURE: 0,            // allow selection of still pictures only. DEFAULT. Will return format specified via DestinationType
    VIDEO: 1,           // allow selection of video only, ONLY RETURNS URL
    ALLMEDIA : 2        // allow selection from all media types
  },
  PictureSourceType:{
    PHOTOLIBRARY : 0,      // Choose image from picture library (same as SAVEDPHOTOALBUM for Android)
    CAMERA : 1,         // Take picture from camera
    SAVEDPHOTOALBUM : 2  // Choose image from picture library (same as PHOTOLIBRARY for Android)
  },
  PopoverArrowDirection:{
    ARROW_UP : 1,        // matches iOS UIPopoverArrowDirection constants to
specify arrow location on popover
    ARROW_DOWN : 2,
    ARROW_LEFT : 4,
    ARROW_RIGHT : 8,
    ARROW_ANY : 15
  },
  Direction:{
    BACK: 0,
    FRONT: 1
  }
};


});


// file: lib/common/plugin/CameraPopoverHandle.js
define("cordova/plugin/CameraPopoverHandle", function(require, exports,
module) {


var exec = require('cordova/exec');
```

```
/**
 * A handle to an image picker popover.
 */
var CameraPopoverHandle = function() {
    this.setPosition = function(popoverOptions) {
        console.log('CameraPopoverHandle.setPosition is only supported on iOS.');
    };
};

module.exports = CameraPopoverHandle;


});


// file: lib/common/plugin/CameraPopoverOptions.js
define("cordova/plugin/CameraPopoverOptions", function(require, exports, module) {

var Camera = require('cordova/plugin/CameraConstants');
```

```
/**
 * Encapsulates options for iOS Popover image picker
 */
var CameraPopoverOptions = function(x,y,width,height,arrowDir){
   // information of rectangle that popover should be anchored to
   this.x = x || 0;
   this.y = y || 32;
   this.width = width || 320;
   this.height = height || 480;
   // The direction of the popover arrow
   this.arrowDir = arrowDir || Camera.PopoverArrowDirection.ARROW_ANY;
};

module.exports = CameraPopoverOptions;

});

// file: lib/common/plugin/CaptureAudioOptions.js
define("cordova/plugin/CaptureAudioOptions", function(require, exports, module) {
```

```
/**
 * Encapsulates all audio capture operation configuration options.
 */
var CaptureAudioOptions = function(){
   // Upper limit of sound clips user can record. Value must be equal or greater than 1.
   this.limit = 1;
   // Maximum duration of a single sound clip in seconds.
   this.duration = 0;
};

module.exports = CaptureAudioOptions;

});

// file: lib/common/plugin/CaptureError.js
define("cordova/plugin/CaptureError", function(require, exports, module) {

/**
```

```
 * The CaptureError interface encapsulates all errors in the Capture API.
 */
var CaptureError = function(c) {
  this.code = c || null;
};

// Camera or microphone failed to capture image or sound.
CaptureError.CAPTURE_INTERNAL_ERR = 0;
// Camera application or audio capture application is currently serving other capture request.
CaptureError.CAPTURE_APPLICATION_BUSY = 1;
// Invalid use of the API (e.g. limit parameter has value less than one).
CaptureError.CAPTURE_INVALID_ARGUMENT = 2;
// User exited camera application or audio capture application before capturing anything.
CaptureError.CAPTURE_NO_MEDIA_FILES = 3;
// The requested capture operation is not supported.
CaptureError.CAPTURE_NOT_SUPPORTED = 20;

module.exports = CaptureError;
```

```
});


// file: lib/common/plugin/CaptureImageOptions.js

define("cordova/plugin/CaptureImageOptions", function(require, exports, module) {


/**

 * Encapsulates all image capture operation configuration options.

 */

var CaptureImageOptions = function(){

    // Upper limit of images user can take. Value must be equal or greater than 1.

    this.limit = 1;

};


module.exports = CaptureImageOptions;


});


// file: lib/common/plugin/CaptureVideoOptions.js
```

```
define("cordova/plugin/CaptureVideoOptions", function(require, exports, module) {

/**
 * Encapsulates all video capture operation configuration options.
 */
var CaptureVideoOptions = function(){
   // Upper limit of videos user can record. Value must be equal or greater than 1.
   this.limit = 1;
   // Maximum duration of a single video clip in seconds.
   this.duration = 0;
};

module.exports = CaptureVideoOptions;

});

// file: lib/common/plugin/CompassError.js
define("cordova/plugin/CompassError", function(require, exports, module) {
```

```
/**
 *  CompassError.
 *  An error code assigned by an implementation when an error has occurred
 * @constructor
 */
var CompassError = function(err) {
    this.code = (err !== undefined ? err : null);
};

CompassError.COMPASS_INTERNAL_ERR = 0;
CompassError.COMPASS_NOT_SUPPORTED = 20;

module.exports = CompassError;


});


// file: lib/common/plugin/CompassHeading.js
define("cordova/plugin/CompassHeading", function(require, exports, module) {
```

```
var CompassHeading = function(magneticHeading, trueHeading, headingAccuracy, timestamp) {
  this.magneticHeading = magneticHeading;
  this.trueHeading = trueHeading;
  this.headingAccuracy = headingAccuracy;
  this.timestamp = timestamp || new Date().getTime();
};

module.exports = CompassHeading;

});

// file: lib/common/plugin/ConfigurationData.js
define("cordova/plugin/ConfigurationData", function(require, exports, module) {

/**
 * Encapsulates a set of parameters that the capture device supports.
 */
function ConfigurationData() {
    // The ASCII-encoded string in lower case representing the media type.
    this.type = null;
    // The height attribute represents height of the image or video in pixels.
    // In the case of a sound clip this attribute has value 0.
    this.height = 0;
    // The width attribute represents width of the image or video in pixels.
    // In the case of a sound clip this attribute has value 0
    this.width = 0;
}

module.exports = ConfigurationData;


});


// file: lib/common/plugin/Connection.js
define("cordova/plugin/Connection", function(require, exports, module) {

/**
 * Network status
 */
```

```
module.exports = {
        UNKNOWN: "unknown",
        ETHERNET: "ethernet",
        WIFI: "wifi",
        CELL_2G: "2g",
        CELL_3G: "3g",
        CELL_4G: "4g",
        CELL:"cellular",
        NONE: "none"
};

});

// file: lib/common/plugin/Contact.js
define("cordova/plugin/Contact", function(require, exports, module) {

var argscheck = require('cordova/argscheck'),
    exec = require('cordova/exec'),
    ContactError = require('cordova/plugin/ContactError'),
    utils = require('cordova/utils');


/**

* Converts primitives into Complex Object

* Currently only used for Date fields

*/

function convertIn(contact) {

    var value = contact.birthday;

    try {

      contact.birthday = new Date(parseFloat(value));

    } catch (exception){

      console.log("Cordova Contact convertIn error: exception creating date.");

    }

    return contact;

}


/**

* Converts Complex objects into primitives

* Only conversion at present is for Dates.
```

```
**/

function convertOut(contact) {
    var value = contact.birthday;
    if (value !== null) {
        // try to make it a Date object if it is not already
        if (!utils.isDate(value)){
            try {
                value = new Date(value);
            } catch(exception){
                value = null;
            }
        }
        if (utils.isDate(value)){
            value = value.valueOf(); // convert to milliseconds
        }
        contact.birthday = value;
    }
    return contact;
}

/**
* Contains information about a single contact.
* @constructor
* @param {DOMString} id unique identifier
* @param {DOMString} displayName
* @param {ContactName} name
* @param {DOMString} nickname
* @param {Array.<ContactField>} phoneNumbers array of phone numbers
* @param {Array.<ContactField>} emails array of email addresses
* @param {Array.<ContactAddress>} addresses array of addresses
* @param {Array.<ContactField>} ims instant messaging user ids
* @param {Array.<ContactOrganization>} organizations
* @param {DOMString} birthday contact's birthday
* @param {DOMString} note user notes about contact
* @param {Array.<ContactField>} photos
* @param {Array.<ContactField>} categories
* @param {Array.<ContactField>} urls contact's web sites
```

```
*/

var Contact = function (id, displayName, name, nickname, phoneNumbers, emails, addresses,

    ims, organizations, birthday, note, photos, categories, urls) {

    this.id = id || null;

    this.rawId = null;

    this.displayName = displayName || null;

    this.name = name || null; // ContactName

    this.nickname = nickname || null;

    this.phoneNumbers = phoneNumbers || null; // ContactField[]

    this.emails = emails || null; // ContactField[]

    this.addresses = addresses || null; // ContactAddress[]

    this.ims = ims || null; // ContactField[]

    this.organizations = organizations || null; // ContactOrganization[]

    this.birthday = birthday || null;

    this.note = note || null;

    this.photos = photos || null; // ContactField[]

    this.categories = categories || null; // ContactField[]

    this.urls = urls || null; // ContactField[]

};
```

```
/**
* Removes contact from device storage.
* @param successCB success callback
* @param errorCB error callback
*/
Contact.prototype.remove = function(successCB, errorCB) {
    argscheck.checkArgs('FF', 'Contact.remove', arguments);
    var fail = errorCB && function(code) {
        errorCB(new ContactError(code));
    };
    if (this.id === null) {
        fail(ContactError.UNKNOWN_ERROR);
    }
    else {
        exec(successCB, fail, "Contacts", "remove", [this.id]);
    }
};
```

```
/**
* Creates a deep copy of this Contact.
* With the contact ID set to null.
* @return copy of this Contact
*/
Contact.prototype.clone = function() {
    var clonedContact = utils.clone(this);
    clonedContact.id = null;
    clonedContact.rawId = null;

    function nullIds(arr) {
        if (arr) {
            for (var i = 0; i < arr.length; ++i) {
                arr[i].id = null;
            }
        }
    }

    // Loop through and clear out any id's in phones, emails, etc.
    nullIds(clonedContact.phoneNumbers);

    nullIds(clonedContact.emails);

    nullIds(clonedContact.addresses);

    nullIds(clonedContact.ims);

    nullIds(clonedContact.organizations);

    nullIds(clonedContact.categories);

    nullIds(clonedContact.photos);

    nullIds(clonedContact.urls);

    return clonedContact;

};


/**

* Persists contact to device storage.

* @param successCB success callback

* @param errorCB error callback

*/

Contact.prototype.save = function(successCB, errorCB) {

    argscheck.checkArgs('FFO', 'Contact.save', arguments);

    var fail = errorCB && function(code) {
        errorCB(new ContactError(code));
    };
    var success = function(result) {
        if (result) {
            if (successCB) {
                var fullContact = require('cordova/plugin/contacts').create(result);
                successCB(convertIn(fullContact));
            }
        }
        else {
            // no Entry object returned
            fail(ContactError.UNKNOWN_ERROR);
        }
    };
    var dupContact = convertOut(utils.clone(this));
    exec(success, fail, "Contacts", "save", [dupContact]);
};
```

```
module.exports = Contact;

});

// file: lib/common/plugin/ContactAddress.js
define("cordova/plugin/ContactAddress", function(require, exports, module) {

/**
* Contact address.
* @constructor
* @param {DOMString} id unique identifier, should only be set by native code
* @param formatted // NOTE: not a W3C standard
* @param streetAddress
* @param locality
* @param region
* @param postalCode
* @param country
*/
```

```
var ContactAddress = function(pref, type, formatted, streetAddress, locality, region, postalCode, country) {
    this.id = null;
    this.pref = (typeof pref != 'undefined' ? pref : false);
    this.type = type || null;
    this.formatted = formatted || null;
    this.streetAddress = streetAddress || null;
    this.locality = locality || null;
    this.region = region || null;
    this.postalCode = postalCode || null;
    this.country = country || null;
};

module.exports = ContactAddress;

});

// file: lib/common/plugin/ContactError.js
define("cordova/plugin/ContactError", function(require, exports, module) {
```

```
/**
 *  ContactError.
 *  An error code assigned by an implementation when an error has occurred
 * @constructor
 */
var ContactError = function(err) {
    this.code = (typeof err != 'undefined' ? err : null);
};

/**
 * Error codes
 */
ContactError.UNKNOWN_ERROR = 0;
ContactError.INVALID_ARGUMENT_ERROR = 1;
ContactError.TIMEOUT_ERROR = 2;
ContactError.PENDING_OPERATION_ERROR = 3;
ContactError.IO_ERROR = 4;
ContactError.NOT_SUPPORTED_ERROR = 5;
ContactError.PERMISSION_DENIED_ERROR = 20;
```

```
module.exports = ContactError;

});

// file: lib/common/plugin/ContactField.js
define("cordova/plugin/ContactField", function(require, exports, module) {

/**
* Generic contact field.
* @constructor
* @param {DOMString} id unique identifier, should only be set by native code // NOTE: not a W3C standard
* @param type
* @param value
* @param pref
*/
var ContactField = function(type, value, pref) {
    this.id = null;
    this.type = (type && type.toString()) || null;
    this.value = (value && value.toString()) || null;

    this.pref = (typeof pref != 'undefined' ? pref : false);

};


module.exports = ContactField;



});



// file: lib/common/plugin/ContactFindOptions.js

define("cordova/plugin/ContactFindOptions", function(require, exports, module)
{



/**

 * ContactFindOptions.

 * @constructor

 * @param filter used to match contacts against

 * @param multiple boolean used to determine if more than one contact should be
returned

 */
```

```
var ContactFindOptions = function(filter, multiple) {
    this.filter = filter || '';
    this.multiple = (typeof multiple != 'undefined' ? multiple : false);
};

module.exports = ContactFindOptions;


});


// file: lib/common/plugin/ContactName.js
define("cordova/plugin/ContactName", function(require, exports, module) {

/**
* Contact name.
* @constructor
* @param formatted // NOTE: not part of W3C standard
* @param familyName
* @param givenName
* @param middle
* @param prefix

* @param suffix

*/

var ContactName = function(formatted, familyName, givenName, middle, prefix, suffix) {

    this.formatted = formatted || null;

    this.familyName = familyName || null;

    this.givenName = givenName || null;

    this.middleName = middle || null;

    this.honorificPrefix = prefix || null;

    this.honorificSuffix = suffix || null;

};


module.exports = ContactName;


});


// file: lib/common/plugin/ContactOrganization.js

define("cordova/plugin/ContactOrganization", function(require, exports, module) {
```

```
/**

* Contact organization.

* @constructor

* @param {DOMString} id unique identifier, should only be set by native code //
NOTE: not a W3C standard

* @param name

* @param dept

* @param title

* @param startDate

* @param endDate

* @param location

* @param desc

*/


var ContactOrganization = function(pref, type, name, dept, title) {

    this.id = null;

    this.pref = (typeof pref != 'undefined' ? pref : false);

    this.type = type || null;

    this.name = name || null;
    this.department = dept || null;
    this.title = title || null;
};

module.exports = ContactOrganization;

});

// file: lib/common/plugin/Coordinates.js
define("cordova/plugin/Coordinates", function(require, exports, module) {

/**
 * This class contains position information.
 * @param {Object} lat
 * @param {Object} lng
 * @param {Object} alt
 * @param {Object} acc
 * @param {Object} head
 * @param {Object} vel
 * @param {Object} altacc
 * @constructor
 */
var Coordinates = function(lat, lng, alt, acc, head, vel, altacc) {
   /**
    * The latitude of the position.
    */
   this.latitude = lat;
   /**
    * The longitude of the position,
    */
   this.longitude = lng;
   /**
    * The accuracy of the position.
    */
   this.accuracy = acc;
   /**
    * The altitude of the position.
    */
   this.altitude = (alt !== undefined ? alt : null);
   /**
    * The direction the device is moving at the position.
    */
   this.heading = (head !== undefined ? head : null);
   /**
    * The velocity with which the device is moving at the position.
    */
   this.speed = (vel !== undefined ? vel : null);

   if (this.speed === 0 || this.speed === null) {
       this.heading = NaN;
   }

   /**
    * The altitude accuracy of the position.
    */
   this.altitudeAccuracy = (altacc !== undefined) ? altacc : null;
};
```

```
module.exports = Coordinates;

});

// file: lib/common/plugin/DirectoryEntry.js
define("cordova/plugin/DirectoryEntry", function(require, exports, module) {

var argscheck = require('cordova/argscheck'),
    utils = require('cordova/utils'),
    exec = require('cordova/exec'),
    Entry = require('cordova/plugin/Entry'),
    FileError = require('cordova/plugin/FileError'),
    DirectoryReader = require('cordova/plugin/DirectoryReader');

/**
 * An interface representing a directory on the file system.
 *
 * {boolean} isFile always false (readonly)
 * {boolean} isDirectory always true (readonly)
 * {DOMString} name of the directory, excluding the path leading to it (readonly)
 * {DOMString} fullPath the absolute full path to the directory (readonly)
 * TODO: implement this!!! {FileSystem} filesystem on which the directory resides (readonly)
 */
var DirectoryEntry = function(name, fullPath) {
    DirectoryEntry.__super__.constructor.call(this, false, true, name, fullPath);
};

utils.extend(DirectoryEntry, Entry);

/**
 * Creates a new DirectoryReader to read entries from this directory
 */
DirectoryEntry.prototype.createReader = function() {
   return new DirectoryReader(this.fullPath);
};

/**
```

```
 * Creates or looks up a directory
 *
 * @param {DOMString} path either a relative or absolute path from this
directory in which to look up or create a directory
 * @param {Flags} options to create or exclusively create the directory
 * @param {Function} successCallback is called with the new entry
 * @param {Function} errorCallback is called with a FileError
 */
DirectoryEntry.prototype.getDirectory = function(path, options, successCallback,
errorCallback) {
   argscheck.checkArgs('sOFF', 'DirectoryEntry.getDirectory', arguments);
   var win = successCallback && function(result) {
      var entry = new DirectoryEntry(result.name, result.fullPath);
      successCallback(entry);
   };
   var fail = errorCallback && function(code) {
      errorCallback(new FileError(code));
   };
   exec(win, fail, "File", "getDirectory", [this.fullPath, path, options]);
};
```

```
/**
 * Deletes a directory and all of it's contents
 *
 * @param {Function} successCallback is called with no parameters
 * @param {Function} errorCallback is called with a FileError
 */
DirectoryEntry.prototype.removeRecursively = function(successCallback,
errorCallback) {
    argscheck.checkArgs('FF', 'DirectoryEntry.removeRecursively', arguments);
    var fail = errorCallback && function(code) {
        errorCallback(new FileError(code));
    };
    exec(successCallback, fail, "File", "removeRecursively", [this.fullPath]);
};


/**
 * Creates or looks up a file
 *
 * @param {DOMString} path either a relative or absolute path from this
directory in which to look up or create a file
 * @param {Flags} options to create or exclusively create the file
 * @param {Function} successCallback is called with the new entry
 * @param {Function} errorCallback is called with a FileError
 */
DirectoryEntry.prototype.getFile = function(path, options, successCallback,
errorCallback) {
   argscheck.checkArgs('sOFF', 'DirectoryEntry.getFile', arguments);
   var win = successCallback && function(result) {
      var FileEntry = require('cordova/plugin/FileEntry');
      var entry = new FileEntry(result.name, result.fullPath);
      successCallback(entry);
   };
   var fail = errorCallback && function(code) {
      errorCallback(new FileError(code));
   };
   exec(win, fail, "File", "getFile", [this.fullPath, path, options]);
};
```

```
module.exports = DirectoryEntry;

});

// file: lib/common/plugin/DirectoryReader.js
define("cordova/plugin/DirectoryReader", function(require, exports, module) {

var exec = require('cordova/exec'),
    FileError = require('cordova/plugin/FileError') ;

/**
 * An interface that lists the files and directories in a directory.
 */
function DirectoryReader(path) {
    this.path = path || null;
}

/**
 * Returns a list of entries from a directory.
 *
 * @param {Function} successCallback is called with a list of entries
 * @param {Function} errorCallback is called with a FileError
 */
DirectoryReader.prototype.readEntries = function(successCallback, errorCallback) {
  var win = typeof successCallback !== 'function' ? null : function(result) {
    var retVal = [];
    for (var i=0; i<result.length; i++) {
      var entry = null;
      if (result[i].isDirectory) {
        entry = new (require('cordova/plugin/DirectoryEntry'))();
      }
      else if (result[i].isFile) {
        entry = new (require('cordova/plugin/FileEntry'))();
      }
      entry.isDirectory = result[i].isDirectory;
      entry.isFile = result[i].isFile;
      entry.name = result[i].name;
      entry.fullPath = result[i].fullPath;
```

```
            retVal.push(entry);
        }
        successCallback(retVal);
    };
    var fail = typeof errorCallback !== 'function' ? null : function(code) {
        errorCallback(new FileError(code));
    };
    exec(win, fail, "File", "readEntries", [this.path]);
};

module.exports = DirectoryReader;

});

// file: lib/common/plugin/Entry.js
define("cordova/plugin/Entry", function(require, exports, module) {

var argscheck = require('cordova/argscheck'),
    exec = require('cordova/exec'),
    FileError = require('cordova/plugin/FileError'),
    Metadata = require('cordova/plugin/Metadata');

/**
 * Represents a file or directory on the local file system.
 *
 * @param isFile
 *            {boolean} true if Entry is a file (readonly)
 * @param isDirectory
 *            {boolean} true if Entry is a directory (readonly)
 * @param name
 *            {DOMString} name of the file or directory, excluding the path
 *            leading to it (readonly)
 * @param fullPath
 *            {DOMString} the absolute full path to the file or directory
 *            (readonly)
 */
function Entry(isFile, isDirectory, name, fullPath, fileSystem) {
    this.isFile = !!isFile;
    this.isDirectory = !!isDirectory;
    this.name = name || '';
    this.fullPath = fullPath || '';
    this.filesystem = fileSystem || null;
}

/**
 * Look up the metadata of the entry.
 *
 * @param successCallback
 *            {Function} is called with a Metadata object
 * @param errorCallback
 *            {Function} is called with a FileError
 */
Entry.prototype.getMetadata = function(successCallback, errorCallback) {
    argscheck.checkArgs('FF', 'Entry.getMetadata', arguments);
    var success = successCallback && function(lastModified) {
        var metadata = new Metadata(lastModified);
        successCallback(metadata);
    };
    var fail = errorCallback && function(code) {
        errorCallback(new FileError(code));
    };

    exec(success, fail, "File", "getMetadata", [this.fullPath]);
};

/**
 * Set the metadata of the entry.
 *
 * @param successCallback
 *          {Function} is called with a Metadata object
 * @param errorCallback
 *          {Function} is called with a FileError
 * @param metadataObject
 *          {Object} keys and values to set
 */
Entry.prototype.setMetadata = function(successCallback, errorCallback,
metadataObject) {
   argscheck.checkArgs('FFO', 'Entry.setMetadata', arguments);
   exec(successCallback, errorCallback, "File", "setMetadata", [this.fullPath, metadataObject]);
};

/**
 * Move a file or directory to a new location.
 *
 * @param parent
 *            {DirectoryEntry} the directory to which to move this entry
 * @param newName
 *            {DOMString} new name of the entry, defaults to the current name
 * @param successCallback
 *            {Function} called with the new DirectoryEntry object
 * @param errorCallback
 *            {Function} called with a FileError
 */
Entry.prototype.moveTo = function(parent, newName, successCallback, errorCallback) {
   argscheck.checkArgs('oSFF', 'Entry.moveTo', arguments);
```

```
var fail = errorCallback && function(code) {
    errorCallback(new FileError(code));
};
// source path
var srcPath = this.fullPath,
    // entry name
    name = newName || this.name,
    success = function(entry) {
        if (entry) {
            if (successCallback) {
                // create appropriate Entry object
                var result = (entry.isDirectory) ? new (require('cordova/plugin/DirectoryEntry'))(entry.name, entry.fullPath) : new (require('cordova/plugin/FileEntry'))(entry.name, entry.fullPath);
                successCallback(result);
            }
        }
        else {
            // no Entry object returned
            fail && fail(FileError.NOT_FOUND_ERR);
            }
        };

    // copy
    exec(success, fail, "File", "moveTo", [srcPath, parent.fullPath, name]);
};

/**
 * Copy a directory to a different location.
 *
 * @param parent
 *            {DirectoryEntry} the directory to which to copy the entry
 * @param newName
 *            {DOMString} new name of the entry, defaults to the current name
 * @param successCallback
 *            {Function} called with the new Entry object
 * @param errorCallback
 *            {Function} called with a FileError
 */
```

```
Entry.prototype.copyTo = function(parent, newName, successCallback,
errorCallback) {
    argscheck.checkArgs('oSFF', 'Entry.copyTo', arguments);
    var fail = errorCallback && function(code) {
        errorCallback(new FileError(code));
    };

        // source path
    var srcPath = this.fullPath,
        // entry name
        name = newName || this.name,
        // success callback
        success = function(entry) {
            if (entry) {
                if (successCallback) {
                    // create appropriate Entry object
                    var result = (entry.isDirectory) ? new
(require('cordova/plugin/DirectoryEntry'))(entry.name, entry.fullPath) : new
(require('cordova/plugin/FileEntry'))(entry.name, entry.fullPath);
                    successCallback(result);
                }
            }
            else {
                // no Entry object returned
                fail && fail(FileError.NOT_FOUND_ERR);
            }
        };

    // copy
    exec(success, fail, "File", "copyTo", [srcPath, parent.fullPath, name]);
};

/**
 * Return a URL that can be used to identify this entry.
 */
Entry.prototype.toURL = function() {
    // fullPath attribute contains the full URL
    return this.fullPath;
};
```

```
/**
 * Returns a URI that can be used to identify this entry.
 *
 * @param {DOMString} mimeType for a FileEntry, the mime type to be used to interpret the file, when loaded through this URI.
 * @return uri
 */
Entry.prototype.toURI = function(mimeType) {
    console.log("DEPRECATED: Update your code to use 'toURL'");
    // fullPath attribute contains the full URI
    return this.toURL();
};


/**
 * Remove a file or directory. It is an error to attempt to delete a
 * directory that is not empty. It is an error to attempt to delete a
 * root directory of a file system.
 *
 * @param successCallback {Function} called with no parameters
 * @param errorCallback {Function} called with a FileError
 */
Entry.prototype.remove = function(successCallback, errorCallback) {
    argscheck.checkArgs('FF', 'Entry.remove', arguments);
    var fail = errorCallback && function(code) {
        errorCallback(new FileError(code));
    };
    exec(successCallback, fail, "File", "remove", [this.fullPath]);
};

/**
 * Look up the parent DirectoryEntry of this entry.
 *
 * @param successCallback {Function} called with the parent DirectoryEntry object
 * @param errorCallback {Function} called with a FileError
 */
Entry.prototype.getParent = function(successCallback, errorCallback) {
    argscheck.checkArgs('FF', 'Entry.getParent', arguments);
    var win = successCallback && function(result) {
        var DirectoryEntry = require('cordova/plugin/DirectoryEntry');
        var entry = new DirectoryEntry(result.name, result.fullPath);
        successCallback(entry);
    };
    var fail = errorCallback && function(code) {
        errorCallback(new FileError(code));
    };
    exec(win, fail, "File", "getParent", [this.fullPath]);
};

module.exports = Entry;


});


// file: lib/common/plugin/File.js
define("cordova/plugin/File", function(require, exports, module) {

/**
 * Constructor.
 * name {DOMString} name of the file, without path information
 * fullPath {DOMString} the full path of the file, including the name
 * type {DOMString} mime type
 * lastModifiedDate {Date} last modified date
 * size {Number} size of the file in bytes
 */

var File = function(name, fullPath, type, lastModifiedDate, size){
   this.name = name || '';
   this.fullPath = fullPath || null;
   this.type = type || null;
   this.lastModifiedDate = lastModifiedDate || null;
   this.size = size || 0;

   // These store the absolute start and end for slicing the file.
   this.start = 0;
   this.end = this.size;
};
```

```
/**
 * Returns a "slice" of the file. Since Cordova Files don't contain the actual
 * content, this really returns a File with adjusted start and end.
 * Slices of slices are supported.
 * start {Number} The index at which to start the slice (inclusive).
 * end {Number} The index at which to end the slice (exclusive).
 */
File.prototype.slice = function(start, end) {
   var size = this.end - this.start;
   var newStart = 0;
   var newEnd = size;
   if (arguments.length) {
      if (start < 0) {
         newStart = Math.max(size + start, 0);
      } else {
         newStart = Math.min(size, start);
      }
   }
```

```
    if (arguments.length >= 2) {
        if (end < 0) {
            newEnd = Math.max(size + end, 0);
        } else {
            newEnd = Math.min(end, size);
        }
    }

    var newFile = new File(this.name, this.fullPath, this.type, this.lastModifiedData, this.size);
    newFile.start = this.start + newStart;
    newFile.end = this.start + newEnd;
    return newFile;
};


module.exports = File;


});
```

```
// file: lib/common/plugin/FileEntry.js
define("cordova/plugin/FileEntry", function(require, exports, module) {


var utils = require('cordova/utils'),
    exec = require('cordova/exec'),
    Entry = require('cordova/plugin/Entry'),
    FileWriter = require('cordova/plugin/FileWriter'),
    File = require('cordova/plugin/File'),
    FileError = require('cordova/plugin/FileError');


/**
 * An interface representing a file on the file system.
 *
 * {boolean} isFile always true (readonly)
 * {boolean} isDirectory always false (readonly)
 * {DOMString} name of the file, excluding the path leading to it (readonly)
 * {DOMString} fullPath the absolute full path to the file (readonly)
 * {FileSystem} filesystem on which the file resides (readonly)
 */
```

```
var FileEntry = function(name, fullPath) {
    FileEntry.__super__.constructor.apply(this, [true, false, name, fullPath]);
};

utils.extend(FileEntry, Entry);

/**
 * Creates a new FileWriter associated with the file that this FileEntry represents.
 *
 * @param {Function} successCallback is called with the new FileWriter
 * @param {Function} errorCallback is called with a FileError
 */
FileEntry.prototype.createWriter = function(successCallback, errorCallback) {
   this.file(function(filePointer) {
      var writer = new FileWriter(filePointer);

      if (writer.fileName === null || writer.fileName === "") {
         errorCallback && errorCallback(new
FileError(FileError.INVALID_STATE_ERR));
      } else {
            successCallback && successCallback(writer);
        }
    }, errorCallback);
};


/**
 * Returns a File that represents the current state of the file that this FileEntry represents.
 *
 * @param {Function} successCallback is called with the new File object
 * @param {Function} errorCallback is called with a FileError
 */
FileEntry.prototype.file = function(successCallback, errorCallback) {
    var win = successCallback && function(f) {
        var file = new File(f.name, f.fullPath, f.type, f.lastModifiedDate, f.size);
        successCallback(file);
    };
    var fail = errorCallback && function(code) {
        errorCallback(new FileError(code));
    };
    exec(win, fail, "File", "getFileMetadata", [this.fullPath]);
};


module.exports = FileEntry;


});


// file: lib/common/plugin/FileError.js
define("cordova/plugin/FileError", function(require, exports, module) {

/**
 * FileError
 */
function FileError(error) {
  this.code = error || null;
}

// File error codes
// Found in DOMException

FileError.NOT_FOUND_ERR = 1;

FileError.SECURITY_ERR = 2;

FileError.ABORT_ERR = 3;


// Added by File API specification

FileError.NOT_READABLE_ERR = 4;

FileError.ENCODING_ERR = 5;

FileError.NO_MODIFICATION_ALLOWED_ERR = 6;

FileError.INVALID_STATE_ERR = 7;

FileError.SYNTAX_ERR = 8;

FileError.INVALID_MODIFICATION_ERR = 9;

FileError.QUOTA_EXCEEDED_ERR = 10;

FileError.TYPE_MISMATCH_ERR = 11;

FileError.PATH_EXISTS_ERR = 12;


module.exports = FileError;


});
```

```
// file: lib/common/plugin/FileReader.js

define("cordova/plugin/FileReader", function(require, exports, module) {


var exec = require('cordova/exec'),

    modulemapper = require('cordova/modulemapper'),

    utils = require('cordova/utils'),

    File = require('cordova/plugin/File'),

    FileError = require('cordova/plugin/FileError'),

    ProgressEvent = require('cordova/plugin/ProgressEvent'),

    origFileReader = modulemapper.getOriginalSymbol(this, 'FileReader');


/**

 * This class reads the mobile device file system.

 *

 * For Android:

 *      The root directory is the root of the file system.

 *      To read from the SD card, the file name is "sdcard/my_file.txt"

 * @constructor
 */
var FileReader = function() {
    this._readyState = 0;
    this._error = null;
    this._result = null;
    this._fileName = '';
    this._realReader = origFileReader ? new origFileReader() : {};
};

// States
FileReader.EMPTY = 0;
FileReader.LOADING = 1;
FileReader.DONE = 2;

utils.defineGetter(FileReader.prototype, 'readyState', function() {
    return this._fileName ? this._readyState : this._realReader.readyState;
});

utils.defineGetter(FileReader.prototype, 'error', function() {
  return this._fileName ? this._error: this._realReader.error;
});

utils.defineGetter(FileReader.prototype, 'result', function() {
  return this._fileName ? this._result: this._realReader.result;
});

function defineEvent(eventName) {
  utils.defineGetterSetter(FileReader.prototype, eventName, function() {
    return this._realReader[eventName] || null;
  }, function(value) {
    this._realReader[eventName] = value;
  });
}
defineEvent('onloadstart');    // When the read starts.
defineEvent('onprogress');     // While reading (and decoding) file or fileBlob data,
and reporting partial file data (progress.loaded/progress.total)
defineEvent('onload');         // When the read has successfully completed.
defineEvent('onerror');        // When the read has failed (see errors).
defineEvent('onloadend');      // When the request has completed (either in success or failure).

defineEvent('onabort');        // When the read has been aborted. For instance, by invoking the abort() method.


function initRead(reader, file) {
  // Already loading something
  if (reader.readyState == FileReader.LOADING) {
   throw new FileError(FileError.INVALID_STATE_ERR);
  }

  reader._result = null;
  reader._error = null;
  reader._readyState = FileReader.LOADING;

  if (typeof file.fullPath == 'string') {
    reader._fileName = file.fullPath;
  } else {
    reader._fileName = '';
    return true;
    }

    reader.onloadstart && reader.onloadstart(new ProgressEvent("loadstart",
{target:reader}));
}

/**
 * Abort reading file.
 */
FileReader.prototype.abort = function() {
    if (origFileReader && !this._fileName) {
        return this._realReader.abort();
    }
    this._result = null;

    if (this._readyState == FileReader.DONE || this._readyState ==
FileReader.EMPTY) {
      return;
    }
    this._readyState = FileReader.DONE;

    // If abort callback
    if (typeof this.onabort === 'function') {
        this.onabort(new ProgressEvent('abort', {target:this}));
    }
    // If load end callback
    if (typeof this.onloadend === 'function') {
        this.onloadend(new ProgressEvent('loadend', {target:this}));
    }
};

/**
 * Read text file.
 *
 * @param file          {File} File object containing file properties
 * @param encoding      [Optional] (see http://www.iana.org/assignments/character-sets)
 */
FileReader.prototype.readAsText = function(file, encoding) {
```

```
if (initRead(this, file)) {
    return this._realReader.readAsText(file, encoding);
}

// Default encoding is UTF-8
var enc = encoding ? encoding : "UTF-8";
var me = this;
var execArgs = [this._fileName, enc, file.start, file.end];

// Read file
exec(
    // Success callback
    function(r) {
        // If DONE (cancelled), then don't do anything
        if (me._readyState === FileReader.DONE) {
            return;
        }

        // Save result
    me._result = r;

    // If onload callback
    if (typeof me.onload === "function") {
        me.onload(new ProgressEvent("load", {target:me}));
    }

    // DONE state
    me._readyState = FileReader.DONE;

    // If onloadend callback
    if (typeof me.onloadend === "function") {
        me.onloadend(new ProgressEvent("loadend", {target:me}));
    }
},
// Error callback
function(e) {
    // If DONE (cancelled), then don't do anything
    if (me._readyState === FileReader.DONE) {
```

```
    return;
}

// DONE state
me._readyState = FileReader.DONE;

// null result
me._result = null;

// Save error
me._error = new FileError(e);

// If onerror callback
if (typeof me.onerror === "function") {
    me.onerror(new ProgressEvent("error", {target:me}));
}

// If onloadend callback
if (typeof me.onloadend === "function") {
```

```
                me.onloadend(new ProgressEvent("loadend", {target:me}));
            }
        }, "File", "readAsText", execArgs);
};


/**
 * Read file and return data as a base64 encoded data url.
 * A data url is of the form:
 *      data:[<mediatype>][;base64],<data>
 *
 * @param file          {File} File object containing file properties
 */
FileReader.prototype.readAsDataURL = function(file) {
    if (initRead(this, file)) {
        return this._realReader.readAsDataURL(file);
    }

    var me = this;
var execArgs = [this._fileName, file.start, file.end];

// Read file
exec(
    // Success callback
    function(r) {
        // If DONE (cancelled), then don't do anything
        if (me._readyState === FileReader.DONE) {
            return;
        }

        // DONE state
        me._readyState = FileReader.DONE;

        // Save result
        me._result = r;

        // If onload callback
        if (typeof me.onload === "function") {
```

```
        me.onload(new ProgressEvent("load", {target:me}));
    }

    // If onloadend callback
    if (typeof me.onloadend === "function") {
        me.onloadend(new ProgressEvent("loadend", {target:me}));
    }
},
// Error callback
function(e) {
    // If DONE (cancelled), then don't do anything
    if (me._readyState === FileReader.DONE) {
        return;
    }

    // DONE state
    me._readyState = FileReader.DONE;

    me._result = null;
```

```
            // Save error
            me._error = new FileError(e);

            // If onerror callback
            if (typeof me.onerror === "function") {
                me.onerror(new ProgressEvent("error", {target:me}));
            }

            // If onloadend callback
            if (typeof me.onloadend === "function") {
                me.onloadend(new ProgressEvent("loadend", {target:me}));
            }
        }, "File", "readAsDataURL", execArgs);
};

/**
 * Read file and return data as a binary data.
 *
 * @param file          {File} File object containing file properties
 */
FileReader.prototype.readAsBinaryString = function(file) {
   if (initRead(this, file)) {
      return this._realReader.readAsBinaryString(file);
   }

   var me = this;
   var execArgs = [this._fileName, file.start, file.end];

   // Read file
   exec(
      // Success callback
      function(r) {
         // If DONE (cancelled), then don't do anything
         if (me._readyState === FileReader.DONE) {
            return;
         }
```

```
        // DONE state
        me._readyState = FileReader.DONE;

        me._result = r;

        // If onload callback
        if (typeof me.onload === "function") {
            me.onload(new ProgressEvent("load", {target:me}));
        }

        // If onloadend callback
        if (typeof me.onloadend === "function") {
            me.onloadend(new ProgressEvent("loadend", {target:me}));
        }
    },
    // Error callback
    function(e) {
        // If DONE (cancelled), then don't do anything
        if (me._readyState === FileReader.DONE) {
```

```
    return;
}

// DONE state
me._readyState = FileReader.DONE;

me._result = null;

// Save error
me._error = new FileError(e);

// If onerror callback
if (typeof me.onerror === "function") {
    me.onerror(new ProgressEvent("error", {target:me}));
}

// If onloadend callback
if (typeof me.onloadend === "function") {
    me.onloadend(new ProgressEvent("loadend", {target:me}));
```

```
            }
        }, "File", "readAsBinaryString", execArgs);
};

/**
 * Read file and return data as a binary data.
 *
 * @param file          {File} File object containing file properties
 */
FileReader.prototype.readAsArrayBuffer = function(file) {
    if (initRead(this, file)) {
        return this._realReader.readAsArrayBuffer(file);
    }

    var me = this;
    var execArgs = [this._fileName, file.start, file.end];

    // Read file
    exec(
// Success callback

function(r) {

  // If DONE (cancelled), then don't do anything

  if (me._readyState === FileReader.DONE) {

    return;

  }


  // DONE state

  me._readyState = FileReader.DONE;


  me._result = r;


  // If onload callback

  if (typeof me.onload === "function") {

    me.onload(new ProgressEvent("load", {target:me}));

  }


  // If onloadend callback

  if (typeof me.onloadend === "function") {
    me.onloadend(new ProgressEvent("loadend", {target:me}));
  }
},
// Error callback
function(e) {
  // If DONE (cancelled), then don't do anything
  if (me._readyState === FileReader.DONE) {
    return;
  }

  // DONE state
  me._readyState = FileReader.DONE;

  me._result = null;

  // Save error
  me._error = new FileError(e);

  // If onerror callback
```

```
            if (typeof me.onerror === "function") {
                me.onerror(new ProgressEvent("error", {target:me}));
            }

            // If onloadend callback
            if (typeof me.onloadend === "function") {
                me.onloadend(new ProgressEvent("loadend", {target:me}));
            }
        }, "File", "readAsArrayBuffer", execArgs);
};

module.exports = FileReader;

});

// file: lib/common/plugin/FileSystem.js
define("cordova/plugin/FileSystem", function(require, exports, module) {

var DirectoryEntry = require('cordova/plugin/DirectoryEntry');
```

```
/**
 * An interface representing a file system
 *
 * @constructor
 * {DOMString} name the unique name of the file system (readonly)
 * {DirectoryEntry} root directory of the file system (readonly)
 */
var FileSystem = function(name, root) {
   this.name = name || null;
   if (root) {
       this.root = new DirectoryEntry(root.name, root.fullPath);
   }
};

module.exports = FileSystem;


});
```

```
// file: lib/common/plugin/FileTransfer.js

define("cordova/plugin/FileTransfer", function(require, exports, module) {


var argscheck = require('cordova/argscheck'),

    exec = require('cordova/exec'),

    FileTransferError = require('cordova/plugin/FileTransferError'),

    ProgressEvent = require('cordova/plugin/ProgressEvent');


function newProgressEvent(result) {

    var pe = new ProgressEvent();

    pe.lengthComputable = result.lengthComputable;

    pe.loaded = result.loaded;

    pe.total = result.total;

    return pe;

}


function getBasicAuthHeader(urlString) {

    var header =  null;
```

```
if (window.btoa) {
    // parse the url using the Location object
    var url = document.createElement('a');
    url.href = urlString;

    var credentials = null;
    var protocol = url.protocol + "//";
    var origin = protocol + url.host;

    // check whether there are the username:password credentials in the url
    if (url.href.indexOf(origin) !== 0) { // credentials found
        var atIndex = url.href.indexOf("@");
        credentials = url.href.substring(protocol.length, atIndex);
    }

    if (credentials) {
        var authHeader = "Authorization";
        var authHeaderValue = "Basic " + window.btoa(credentials);
```

```
            header = {
                name : authHeader,
                value : authHeaderValue
            };
        }
    }

    return header;
}

var idCounter = 0;

/**
 * FileTransfer uploads a file to a remote server.
 * @constructor
 */
var FileTransfer = function() {
    this._id = ++idCounter;
    this.onprogress = null; // optional callback
```

```
};

/**
* Given an absolute file path, uploads a file on the device to a remote server
* using a multipart HTTP request.
* @param filePath {String}          Full path of the file on the device
* @param server {String}            URL of the server to receive the file
* @param successCallback (Function}  Callback to be invoked when upload has
completed
* @param errorCallback {Function}    Callback to be invoked upon error
* @param options {FileUploadOptions} Optional parameters such as file name
and mimetype
* @param trustAllHosts {Boolean} Optional trust all hosts (e.g. for self-signed
certs), defaults to false
*/
FileTransfer.prototype.upload = function(filePath, server, successCallback,
errorCallback, options, trustAllHosts) {
   argscheck.checkArgs('ssFFO*', 'FileTransfer.upload', arguments);
   // check for options
   var fileKey = null;
   var fileName = null;
   var mimeType = null;

   var params = null;

   var chunkedMode = true;

   var headers = null;

   var httpMethod = null;

   var basicAuthHeader = getBasicAuthHeader(server);

   if (basicAuthHeader) {

      options = options || {};

      options.headers = options.headers || {};

      options.headers[basicAuthHeader.name] = basicAuthHeader.value;

   }


   if (options) {

      fileKey = options.fileKey;

      fileName = options.fileName;

      mimeType = options.mimeType;

      headers = options.headers;

      httpMethod = options.httpMethod || "POST";

      if (httpMethod.toUpperCase() == "PUT"){
        httpMethod = "PUT";
      } else {
        httpMethod = "POST";
      }
      if (options.chunkedMode !== null || typeof options.chunkedMode != "undefined") {
        chunkedMode = options.chunkedMode;
      }
      if (options.params) {
        params = options.params;
      }
      else {
        params = {};
      }
   }

   var fail = errorCallback && function(e) {
      var error = new FileTransferError(e.code, e.source, e.target, e.http_status, e.body);
      errorCallback(error);
    };


    var self = this;
    var win = function(result) {
        if (typeof result.lengthComputable != "undefined") {
            if (self.onprogress) {
                self.onprogress(newProgressEvent(result));
            }
        } else {
            successCallback && successCallback(result);
        }
    };
    exec(win, fail, 'FileTransfer', 'upload', [filePath, server, fileKey, fileName,
mimeType, params, trustAllHosts, chunkedMode, headers, this._id, httpMethod]);
};


/**
 * Downloads a file form a given URL and saves it to the specified directory.
 * @param source {String}          URL of the server to receive the file
 * @param target {String}         Full path of the file on the device
 * @param successCallback (Function}  Callback to be invoked when upload has
completed
 * @param errorCallback {Function}    Callback to be invoked upon error
 * @param trustAllHosts {Boolean} Optional trust all hosts (e.g. for self-signed
certs), defaults to false
 * @param options {FileDownloadOptions} Optional parameters such as headers
 */
FileTransfer.prototype.download = function(source, target, successCallback,
errorCallback, trustAllHosts, options) {
    argscheck.checkArgs('ssFF*', 'FileTransfer.download', arguments);
    var self = this;

    var basicAuthHeader = getBasicAuthHeader(source);
    if (basicAuthHeader) {
        options = options || {};
        options.headers = options.headers || {};
        options.headers[basicAuthHeader.name] = basicAuthHeader.value;
    }

    var headers = null;
```

```
if (options) {
    headers = options.headers || null;
}

var win = function(result) {
    if (typeof result.lengthComputable != "undefined") {
        if (self.onprogress) {
            return self.onprogress(newProgressEvent(result));
        }
    } else if (successCallback) {
        var entry = null;
        if (result.isDirectory) {
            entry = new (require('cordova/plugin/DirectoryEntry'))();
        }
        else if (result.isFile) {
            entry = new (require('cordova/plugin/FileEntry'))();
        }
        entry.isDirectory = result.isDirectory;
        entry.isFile = result.isFile;
```

```
            entry.name = result.name;

            entry.fullPath = result.fullPath;

            successCallback(entry);

        }

    };


    var fail = errorCallback && function(e) {

        var error = new FileTransferError(e.code, e.source, e.target, e.http_status, e.body);

        errorCallback(error);

    };


    exec(win, fail, 'FileTransfer', 'download', [source, target, trustAllHosts, this._id, headers]);

};


/**

 * Aborts the ongoing file transfer on this object. The original error

 * callback for the file transfer will be called if necessary.

 */
```

```
FileTransfer.prototype.abort = function() {
    exec(null, null, 'FileTransfer', 'abort', [this._id]);
};

module.exports = FileTransfer;


});


// file: lib/common/plugin/FileTransferError.js
define("cordova/plugin/FileTransferError", function(require, exports, module) {

/**
 * FileTransferError
 * @constructor
 */
var FileTransferError = function(code, source, target, status, body) {
    this.code = code || null;
    this.source = source || null;
    this.target = target || null;
    this.http_status = status || null;
    this.body = body || null;
};

FileTransferError.FILE_NOT_FOUND_ERR = 1;
FileTransferError.INVALID_URL_ERR = 2;
FileTransferError.CONNECTION_ERR = 3;
FileTransferError.ABORT_ERR = 4;

module.exports = FileTransferError;


});


// file: lib/common/plugin/FileUploadOptions.js
define("cordova/plugin/FileUploadOptions", function(require, exports, module) {

/**
 * Options to customize the HTTP request used to upload files.
 * @constructor
 * @param fileKey {String}   Name of file request parameter.
 * @param fileName {String}   Filename to be used by the server. Defaults to image.jpg.
 * @param mimeType {String}   Mimetype of the uploaded file. Defaults to image/jpeg.
 * @param params {Object}      Object with key: value params to send to the server.
 * @param headers {Object}   Keys are header names, values are header values. Multiple
 *                          headers of the same name are not supported.
 */
var FileUploadOptions = function(fileKey, fileName, mimeType, params, headers, httpMethod) {
   this.fileKey = fileKey || null;
   this.fileName = fileName || null;
   this.mimeType = mimeType || null;
   this.params = params || null;
   this.headers = headers || null;
   this.httpMethod = httpMethod || null;
};
```

```
module.exports = FileUploadOptions;

});

// file: lib/common/plugin/FileUploadResult.js
define("cordova/plugin/FileUploadResult", function(require, exports, module) {

/**
 * FileUploadResult
 * @constructor
 */
var FileUploadResult = function() {
    this.bytesSent = 0;
    this.responseCode = null;
    this.response = null;
};

module.exports = FileUploadResult;
```

```
});

// file: lib/common/plugin/FileWriter.js
define("cordova/plugin/FileWriter", function(require, exports, module) {

var exec = require('cordova/exec'),
    FileError = require('cordova/plugin/FileError'),
    ProgressEvent = require('cordova/plugin/ProgressEvent');

/**
 * This class writes to the mobile device file system.
 *
 * For Android:
 *      The root directory is the root of the file system.
 *      To write to the SD card, the file name is "sdcard/my_file.txt"
 *
 * @constructor
 * @param file {File} File object containing file properties
 * @param append if true write to the end of the file, otherwise overwrite the file
 */
var FileWriter = function(file) {
    this.fileName = "";
    this.length = 0;
    if (file) {
        this.fileName = file.fullPath || file;
        this.length = file.size || 0;
    }
    // default is to write at the beginning of the file
    this.position = 0;

    this.readyState = 0; // EMPTY

    this.result = null;

    // Error
    this.error = null;

    // Event handlers
```

```
   this.onwritestart = null;   // When writing starts
   this.onprogress = null;     // While writing the file, and reporting partial file data
   this.onwrite = null;        // When the write has successfully completed.
   this.onwriteend = null;     // When the request has completed (either in success or failure).
   this.onabort = null;        // When the write has been aborted. For instance, by invoking the abort() method.
   this.onerror = null;        // When the write has failed (see errors).
};


// States
FileWriter.INIT = 0;
FileWriter.WRITING = 1;
FileWriter.DONE = 2;


/**
 * Abort writing file.
 */
FileWriter.prototype.abort = function() {
   // check for invalid state
```

```
    if (this.readyState === FileWriter.DONE || this.readyState === FileWriter.INIT) {
        throw new FileError(FileError.INVALID_STATE_ERR);
    }

    // set error
    this.error = new FileError(FileError.ABORT_ERR);

    this.readyState = FileWriter.DONE;

    // If abort callback
    if (typeof this.onabort === "function") {
        this.onabort(new ProgressEvent("abort", {"target":this}));
    }

    // If write end callback
    if (typeof this.onwriteend === "function") {
        this.onwriteend(new ProgressEvent("writeend", {"target":this}));
    }
};
```

```
/**
 * Writes data to the file
 *
 * @param data text or blob to be written
 */
FileWriter.prototype.write = function(data) {

   var isBinary = false;

   // If we don't have Blob or ArrayBuffer support, don't bother.
   if (typeof window.Blob !== 'undefined' && typeof window.ArrayBuffer !==
'undefined') {

       // Check to see if the incoming data is a blob
       if (data instanceof Blob) {
           var that=this;
           var fileReader = new FileReader();
           fileReader.onload = function() {
               // Call this method again, with the arraybuffer as argument
```

```
      FileWriter.prototype.write.call(that, this.result);
    };
    fileReader.readAsArrayBuffer(data);
    return;
  }

  // Mark data type for safer transport over the binary bridge
  isBinary = (data instanceof ArrayBuffer);
}

// Throw an exception if we are already writing a file
if (this.readyState === FileWriter.WRITING) {
  throw new FileError(FileError.INVALID_STATE_ERR);
}

// WRITING state
this.readyState = FileWriter.WRITING;

var me = this;
```

```
// If onwritestart callback
if (typeof me.onwritestart === "function") {
    me.onwritestart(new ProgressEvent("writestart", {"target":me}));
}

// Write file
exec(
    // Success callback
    function(r) {
        // If DONE (cancelled), then don't do anything
        if (me.readyState === FileWriter.DONE) {
            return;
        }

        // position always increases by bytes written because file would be extended
        me.position += r;
        // The length of the file is now where we are done writing.
    me.length = me.position;

    // DONE state
    me.readyState = FileWriter.DONE;

    // If onwrite callback
    if (typeof me.onwrite === "function") {
        me.onwrite(new ProgressEvent("write", {"target":me}));
    }

    // If onwriteend callback
    if (typeof me.onwriteend === "function") {
        me.onwriteend(new ProgressEvent("writeend", {"target":me}));
    }
},
// Error callback
function(e) {
    // If DONE (cancelled), then don't do anything
    if (me.readyState === FileWriter.DONE) {
        return;
    }

    // DONE state
    me.readyState = FileWriter.DONE;

    // Save error
    me.error = new FileError(e);

    // If onerror callback
    if (typeof me.onerror === "function") {
        me.onerror(new ProgressEvent("error", {"target":me}));
    }

    // If onwriteend callback
    if (typeof me.onwriteend === "function") {
        me.onwriteend(new ProgressEvent("writeend", {"target":me}));
    }
}, "File", "write", [this.fileName, data, this.position, isBinary]);
```

```
};

/**
 * Moves the file pointer to the location specified.
 *
 * If the offset is a negative number the position of the file
 * pointer is rewound.  If the offset is greater than the file
 * size the position is set to the end of the file.
 *
 * @param offset is the location to move the file pointer to.
 */
FileWriter.prototype.seek = function(offset) {
    // Throw an exception if we are already writing a file
    if (this.readyState === FileWriter.WRITING) {
        throw new FileError(FileError.INVALID_STATE_ERR);
    }

    if (!offset && offset !== 0) {
        return;
    }

    // See back from end of file.
    if (offset < 0) {
        this.position = Math.max(offset + this.length, 0);
    }
    // Offset is bigger than file size so set position
    // to the end of the file.
    else if (offset > this.length) {
        this.position = this.length;
    }
    // Offset is between 0 and file size so set the position
    // to start writing.
    else {
        this.position = offset;
    }
};

/**
```

```
 * Truncates the file to the size specified.
 *
 * @param size to chop the file at.
 */
FileWriter.prototype.truncate = function(size) {
    // Throw an exception if we are already writing a file
    if (this.readyState === FileWriter.WRITING) {
        throw new FileError(FileError.INVALID_STATE_ERR);
    }

    // WRITING state
    this.readyState = FileWriter.WRITING;

    var me = this;

    // If onwritestart callback
    if (typeof me.onwritestart === "function") {
        me.onwritestart(new ProgressEvent("writestart", {"target":this}));
    }
```

```
// Write file
exec(
  // Success callback
  function(r) {
    // If DONE (cancelled), then don't do anything
    if (me.readyState === FileWriter.DONE) {
      return;
    }

    // DONE state
    me.readyState = FileWriter.DONE;

    // Update the length of the file
    me.length = r;
    me.position = Math.min(me.position, r);

    // If onwrite callback
    if (typeof me.onwrite === "function") {
        me.onwrite(new ProgressEvent("write", {"target":me}));
    }

    // If onwriteend callback
    if (typeof me.onwriteend === "function") {
        me.onwriteend(new ProgressEvent("writeend", {"target":me}));
    }
},
// Error callback
function(e) {
    // If DONE (cancelled), then don't do anything
    if (me.readyState === FileWriter.DONE) {
        return;
    }

    // DONE state
    me.readyState = FileWriter.DONE;

    // Save error
```

```
            me.error = new FileError(e);

            // If onerror callback
            if (typeof me.onerror === "function") {
                me.onerror(new ProgressEvent("error", {"target":me}));
            }

            // If onwriteend callback
            if (typeof me.onwriteend === "function") {
                me.onwriteend(new ProgressEvent("writeend", {"target":me}));
            }
        }, "File", "truncate", [this.fileName, size]);
};

module.exports = FileWriter;

});

// file: lib/common/plugin/Flags.js
```

```
define("cordova/plugin/Flags", function(require, exports, module) {

/**
 * Supplies arguments to methods that lookup or create files and directories.
 *
 * @param create
 *           {boolean} file or directory if it doesn't exist
 * @param exclusive
 *           {boolean} used with create; if true the command will fail if
 *           target path exists
 */
function Flags(create, exclusive) {
    this.create = create || false;
    this.exclusive = exclusive || false;
}

module.exports = Flags;

});
```

```
// file: lib/common/plugin/GlobalizationError.js

define("cordova/plugin/GlobalizationError", function(require, exports, module) {



/**
 * Globalization error object
 *
 * @constructor
 * @param code
 * @param message
 */
var GlobalizationError = function(code, message) {
   this.code = code || null;
   this.message = message || '';
};

// Globalization error codes
GlobalizationError.UNKNOWN_ERROR = 0;
GlobalizationError.FORMATTING_ERROR = 1;
GlobalizationError.PARSING_ERROR = 2;
GlobalizationError.PATTERN_ERROR = 3;

module.exports = GlobalizationError;

});

// file: lib/common/plugin/InAppBrowser.js
define("cordova/plugin/InAppBrowser", function(require, exports, module) {

var exec = require('cordova/exec');
var channel = require('cordova/channel');
var modulemapper = require('cordova/modulemapper');

function InAppBrowser() {
   this.channels = {
        'loadstart': channel.create('loadstart'),
        'loadstop' : channel.create('loadstop'),
        'loaderror' : channel.create('loaderror'),
        'exit' : channel.create('exit')
    };
}

InAppBrowser.prototype = {
    _eventHandler: function (event) {
        if (event.type in this.channels) {
            this.channels[event.type].fire(event);
        }
    },
    close: function (eventname) {
        exec(null, null, "InAppBrowser", "close", []);
    },
    show: function (eventname) {
      exec(null, null, "InAppBrowser", "show", []);
    },
    addEventListener: function (eventname,f) {
        if (eventname in this.channels) {
```

```
      this.channels[eventname].subscribe(f);
    }
  },
  removeEventListener: function(eventname, f) {
    if (eventname in this.channels) {
      this.channels[eventname].unsubscribe(f);
    }
  },

  executeScript: function(injectDetails, cb) {
    if (injectDetails.code) {
      exec(cb, null, "InAppBrowser", "injectScriptCode", [injectDetails.code, !!cb]);
    } else if (injectDetails.file) {
      exec(cb, null, "InAppBrowser", "injectScriptFile", [injectDetails.file, !!cb]);
    } else {
      throw new Error('executeScript requires exactly one of code or file to be specified');
    }
    },

    insertCSS: function(injectDetails, cb) {
        if (injectDetails.code) {
            exec(cb, null, "InAppBrowser", "injectStyleCode", [injectDetails.code, !!cb]);
        } else if (injectDetails.file) {
            exec(cb, null, "InAppBrowser", "injectStyleFile", [injectDetails.file, !!cb]);
        } else {
            throw new Error('insertCSS requires exactly one of code or file to be specified');
        }
    }
};

module.exports = function(strUrl, strWindowName, strWindowFeatures) {
    var iab = new InAppBrowser();
    var cb = function(eventname) {
       iab._eventHandler(eventname);
    };

    // Don't catch calls that write to existing frames (e.g. named iframes).
    if (window.frames && window.frames[strWindowName]) {
        var origOpenFunc = modulemapper.getOriginalSymbol(window, 'open');
        return origOpenFunc.apply(window, arguments);
    }

    exec(cb, cb, "InAppBrowser", "open", [strUrl, strWindowName, strWindowFeatures]);
    return iab;
};


});

// file: lib/common/plugin/LocalFileSystem.js
define("cordova/plugin/LocalFileSystem", function(require, exports, module) {

var exec = require('cordova/exec');
```

```
/**
 * Represents a local file system.
 */
var LocalFileSystem = function() {

};

LocalFileSystem.TEMPORARY = 0; //temporary, with no guarantee of persistence
LocalFileSystem.PERSISTENT = 1; //persistent

module.exports = LocalFileSystem;

});

// file: lib/common/plugin/Media.js
define("cordova/plugin/Media", function(require, exports, module) {

var argscheck = require('cordova/argscheck'),
    utils = require('cordova/utils'),
    exec = require('cordova/exec');

var mediaObjects = {};

/**
 * This class provides access to the device media, interfaces to both sound and video
 *
 * @constructor
 * @param src                   The file name or url to play
 * @param successCallback       The callback to be called when the file is done playing or recording.
 *                                  successCallback()
 * @param errorCallback         The callback to be called if there is an error.
 *                                  errorCallback(int errorCode) - OPTIONAL
 * @param statusCallback        The callback to be called when media status has changed.
 *                                  statusCallback(int statusCode) - OPTIONAL
 */
```

```
var Media = function(src, successCallback, errorCallback, statusCallback) {
    argscheck.checkArgs('SFFF', 'Media', arguments);
    this.id = utils.createUUID();
    mediaObjects[this.id] = this;
    this.src = src;
    this.successCallback = successCallback;
    this.errorCallback = errorCallback;
    this.statusCallback = statusCallback;
    this._duration = -1;
    this._position = -1;
    exec(null, this.errorCallback, "Media", "create", [this.id, this.src]);
};

// Media messages
Media.MEDIA_STATE = 1;
Media.MEDIA_DURATION = 2;
Media.MEDIA_POSITION = 3;
Media.MEDIA_ERROR = 9;
```

```
// Media states

Media.MEDIA_NONE = 0;

Media.MEDIA_STARTING = 1;

Media.MEDIA_RUNNING = 2;

Media.MEDIA_PAUSED = 3;

Media.MEDIA_STOPPED = 4;

Media.MEDIA_MSG = ["None", "Starting", "Running", "Paused", "Stopped"];


// "static" function to return existing objs.

Media.get = function(id) {

    return mediaObjects[id];

};


/**

 * Start or resume playing audio file.

 */

Media.prototype.play = function(options) {

    exec(null, null, "Media", "startPlayingAudio", [this.id, this.src, options]);

};
```

```
/**
 * Stop playing audio file.
 */
Media.prototype.stop = function() {
    var me = this;
    exec(function() {
        me._position = 0;
    }, this.errorCallback, "Media", "stopPlayingAudio", [this.id]);
};

/**
 * Seek or jump to a new time in the track..
 */
Media.prototype.seekTo = function(milliseconds) {
    var me = this;
    exec(function(p) {
        me._position = p;
    }, this.errorCallback, "Media", "seekToAudio", [this.id, milliseconds]);
};

/**
 * Pause playing audio file.
 */
Media.prototype.pause = function() {
    exec(null, this.errorCallback, "Media", "pausePlayingAudio", [this.id]);
};

/**
 * Get duration of an audio file.
 * The duration is only set for audio that is playing, paused or stopped.
 *
 * @return      duration or -1 if not known.
 */
Media.prototype.getDuration = function() {
    return this._duration;
};
```

```
/**
 * Get position of audio.
 */
Media.prototype.getCurrentPosition = function(success, fail) {
    var me = this;
    exec(function(p) {
        me._position = p;
        success(p);
    }, fail, "Media", "getCurrentPositionAudio", [this.id]);
};

/**
 * Start recording audio file.
 */
Media.prototype.startRecord = function() {
    exec(null, this.errorCallback, "Media", "startRecordingAudio", [this.id, this.src]);
};

/**
```

```
 * Stop recording audio file.
 */
Media.prototype.stopRecord = function() {
   exec(null, this.errorCallback, "Media", "stopRecordingAudio", [this.id]);
};

/**
 * Release the resources.
 */
Media.prototype.release = function() {
   exec(null, this.errorCallback, "Media", "release", [this.id]);
};

/**
 * Adjust the volume.
 */
Media.prototype.setVolume = function(volume) {
   exec(null, null, "Media", "setVolume", [this.id, volume]);
};
```

```
/**
 * Audio has status update.
 * PRIVATE
 *
 * @param id            The media object id (string)
 * @param msgType       The 'type' of update this is
 * @param value         Use of value is determined by the msgType
 */
Media.onStatus = function(id, msgType, value) {

    var media = mediaObjects[id];

    if(media) {
        switch(msgType) {
            case Media.MEDIA_STATE :
                media.statusCallback && media.statusCallback(value);
                if(value == Media.MEDIA_STOPPED) {
                    media.successCallback && media.successCallback();
                }
                break;
            case Media.MEDIA_DURATION :
                media._duration = value;
                break;
            case Media.MEDIA_ERROR :
                media.errorCallback && media.errorCallback(value);
                break;
            case Media.MEDIA_POSITION :
                media._position = Number(value);
                break;
            default :
                console.error && console.error("Unhandled Media.onStatus :: " + msgType);
                break;
        }
    }
    else {
       console.error && console.error("Received Media.onStatus callback for unknown media :: " + id);
    }

};

module.exports = Media;

});

// file: lib/common/plugin/MediaError.js
define("cordova/plugin/MediaError", function(require, exports, module) {

/**
 * This class contains information about any Media errors.
*/
/*
 According to :: http://dev.w3.org/html5/spec-author-view/video.html#mediaerror
 We should never be creating these objects, we should just implement the interface
 which has 1 property for an instance, 'code'
 instead of doing :

    errorCallbackFunction( new MediaError(3,'msg') );

we should simply use a literal :

    errorCallbackFunction( {'code':3} );

 */


 var _MediaError = window.MediaError;



if(!_MediaError) {

    window.MediaError = _MediaError = function(code, msg) {

        this.code = (typeof code != 'undefined') ? code : null;

        this.message = msg || ""; // message is NON-standard! do not use!

    };

}


_MediaError.MEDIA_ERR_NONE_ACTIVE                                    =
_MediaError.MEDIA_ERR_NONE_ACTIVE    || 0;

_MediaError.MEDIA_ERR_ABORTED                                        =
_MediaError.MEDIA_ERR_ABORTED        || 1;
_MediaError.MEDIA_ERR_NETWORK                                            =
_MediaError.MEDIA_ERR_NETWORK       || 2;
_MediaError.MEDIA_ERR_DECODE                                             =
_MediaError.MEDIA_ERR_DECODE        || 3;
_MediaError.MEDIA_ERR_NONE_SUPPORTED                                     =
_MediaError.MEDIA_ERR_NONE_SUPPORTED || 4;
// TODO: MediaError.MEDIA_ERR_NONE_SUPPORTED is legacy, the W3
spec now defines it as below.
// as defined by http://dev.w3.org/html5/spec-author-view/video.html#error-codes
_MediaError.MEDIA_ERR_SRC_NOT_SUPPORTED                                  =
_MediaError.MEDIA_ERR_SRC_NOT_SUPPORTED || 4;


module.exports = _MediaError;


});


// file: lib/common/plugin/MediaFile.js
define("cordova/plugin/MediaFile", function(require, exports, module) {


var utils = require('cordova/utils'),
    exec = require('cordova/exec'),
    File = require('cordova/plugin/File'),
    CaptureError = require('cordova/plugin/CaptureError');
/**
 * Represents a single file.
 *
 * name {DOMString} name of the file, without path information
 * fullPath {DOMString} the full path of the file, including the name
 * type {DOMString} mime type
 * lastModifiedDate {Date} last modified date
 * size {Number} size of the file in bytes
 */
var MediaFile = function(name, fullPath, type, lastModifiedDate, size){
    MediaFile.__super__.constructor.apply(this, arguments);
};

utils.extend(MediaFile, File);

/**
 * Request capture format data for a specific file and type
 *
 * @param {Function} successCB
 * @param {Function} errorCB
 */
MediaFile.prototype.getFormatData = function(successCallback, errorCallback) {
    if (typeof this.fullPath === "undefined" || this.fullPath === null) {
        errorCallback(new
CaptureError(CaptureError.CAPTURE_INVALID_ARGUMENT));
    } else {
        exec(successCallback, errorCallback, "Capture", "getFormatData",
[this.fullPath, this.type]);
    }
};

module.exports = MediaFile;

});

// file: lib/common/plugin/MediaFileData.js
define("cordova/plugin/MediaFileData", function(require, exports, module) {
```

```
/**
 * MediaFileData encapsulates format information of a media file.
 *
 * @param {DOMString} codecs
 * @param {long} bitrate
 * @param {long} height
 * @param {long} width
 * @param {float} duration
 */
var MediaFileData = function(codecs, bitrate, height, width, duration){
   this.codecs = codecs || null;
   this.bitrate = bitrate || 0;
   this.height = height || 0;
   this.width = width || 0;
   this.duration = duration || 0;
};

module.exports = MediaFileData;
```

```
});


// file: lib/common/plugin/Metadata.js

define("cordova/plugin/Metadata", function(require, exports, module) {


/**
 * Information about the state of the file or directory
 *
 * {Date} modificationTime (readonly)
 */
var Metadata = function(time) {
    this.modificationTime = (typeof time != 'undefined'?new Date(time):null);
};


module.exports = Metadata;


});
```

```
// file: lib/common/plugin/Position.js
define("cordova/plugin/Position", function(require, exports, module) {


var Coordinates = require('cordova/plugin/Coordinates');


var Position = function(coords, timestamp) {
   if (coords) {
      this.coords = new Coordinates(coords.latitude, coords.longitude, coords.altitude, coords.accuracy, coords.heading, coords.velocity, coords.altitudeAccuracy);
   } else {
      this.coords = new Coordinates();
   }
   this.timestamp = (timestamp !== undefined) ? timestamp : new Date();
};


module.exports = Position;


});
```

```
// file: lib/common/plugin/PositionError.js
define("cordova/plugin/PositionError", function(require, exports, module) {

/**
 * Position error object
 *
 * @constructor
 * @param code
 * @param message
 */
var PositionError = function(code, message) {
    this.code = code || null;
    this.message = message || '';
};

PositionError.PERMISSION_DENIED = 1;
PositionError.POSITION_UNAVAILABLE = 2;
PositionError.TIMEOUT = 3;
```

```
module.exports = PositionError;

});

// file: lib/common/plugin/ProgressEvent.js
define("cordova/plugin/ProgressEvent", function(require, exports, module) {

// If ProgressEvent exists in global context, use it already, otherwise use our own polyfill
// Feature test: See if we can instantiate a native ProgressEvent;
// if so, use that approach,
// otherwise fill-in with our own implementation.
//
// NOTE: right now we always fill in with our own. Down the road would be nice if we can use whatever is native in the webview.
var ProgressEvent = (function() {
    /*
    var createEvent = function(data) {
        var event = document.createEvent('Events');
        event.initEvent('ProgressEvent', false, false);
        if (data) {
            for (var i in data) {
                if (data.hasOwnProperty(i)) {
                    event[i] = data[i];
                }
            }
            if (data.target) {
                // TODO: cannot call <some_custom_object>.dispatchEvent
                // need to first figure out how to implement EventTarget
            }
        }
        return event;
    };
    try {
        var ev = createEvent({type:"abort",target:document});
        return function ProgressEvent(type, data) {
            data.type = type;
            return createEvent(data);
        };
    } catch(e){
    */
        return function ProgressEvent(type, dict) {
            this.type = type;
            this.bubbles = false;
            this.cancelBubble = false;
            this.cancelable = false;
            this.lengthComputable = false;
            this.loaded = dict && dict.loaded ? dict.loaded : 0;
            this.total = dict && dict.total ? dict.total : 0;
            this.target = dict && dict.target ? dict.target : null;
        };
    //}
})();

module.exports = ProgressEvent;


});
```

```
// file: lib/common/plugin/accelerometer.js
define("cordova/plugin/accelerometer", function(require, exports, module) {

/**
 * This class provides access to device accelerometer data.
 * @constructor
 */
var argscheck = require('cordova/argscheck'),
    utils = require("cordova/utils"),
    exec = require("cordova/exec"),
    Acceleration = require('cordova/plugin/Acceleration');

// Is the accel sensor running?
var running = false;

// Keeps reference to watchAcceleration calls.
var timers = {};

// Array of listeners; used to keep track of when we should call start and stop.
```

```
var listeners = [];

// Last returned acceleration object from native
var accel = null;

// Tells native to start.
function start() {
    exec(function(a) {
        var tempListeners = listeners.slice(0);
        accel = new Acceleration(a.x, a.y, a.z, a.timestamp);
        for (var i = 0, l = tempListeners.length; i < l; i++) {
            tempListeners[i].win(accel);
        }
    }, function(e) {
        var tempListeners = listeners.slice(0);
        for (var i = 0, l = tempListeners.length; i < l; i++) {
            tempListeners[i].fail(e);
        }
    }, "Accelerometer", "start", []);
    running = true;
}

// Tells native to stop.
function stop() {
    exec(null, null, "Accelerometer", "stop", []);
    running = false;
}

// Adds a callback pair to the listeners array
function createCallbackPair(win, fail) {
    return {win:win, fail:fail};
}

// Removes a win/fail listener pair from the listeners array
function removeListeners(l) {
    var idx = listeners.indexOf(l);
    if (idx > -1) {
        listeners.splice(idx, 1);
        if (listeners.length === 0) {
            stop();
        }
    }
}

var accelerometer = {
    /**
     * Asynchronously acquires the current acceleration.
     *
     * @param {Function} successCallback      The function to call when the acceleration data is available
     * @param {Function} errorCallback      The function to call when there is an error getting the acceleration data. (OPTIONAL)
     * @param {AccelerationOptions} options The options for getting the accelerometer data such as timeout. (OPTIONAL)
     */
    getCurrentAcceleration: function(successCallback, errorCallback, options) {
        argscheck.checkArgs('fFO', 'accelerometer.getCurrentAcceleration', arguments);
        var p;
        var win = function(a) {
            removeListeners(p);
            successCallback(a);
        };
        var fail = function(e) {
            removeListeners(p);
            errorCallback && errorCallback(e);
        };

        p = createCallbackPair(win, fail);
        listeners.push(p);

        if (!running) {
            start();
        }
    },

    /**
```

```
    * Asynchronously acquires the acceleration repeatedly at a given interval.
    *
    * @param {Function} successCallback    The function to call each time the acceleration data is available
    * @param {Function} errorCallback      The function to call when there is an error getting the acceleration data. (OPTIONAL)
    * @param {AccelerationOptions} options The options for getting the accelerometer data such as timeout. (OPTIONAL)
    * @return String                       The watch id that must be passed to #clearWatch to stop watching.
    */
   watchAcceleration: function(successCallback, errorCallback, options) {
      argscheck.checkArgs('fFO', 'accelerometer.watchAcceleration', arguments);
      // Default interval (10 sec)
      var frequency = (options && options.frequency && typeof options.frequency == 'number') ? options.frequency : 10000;

      // Keep reference to watch id, and report accel readings as often as defined in frequency
      var id = utils.createUUID();
```

```
var p = createCallbackPair(function(){}, function(e) {
    removeListeners(p);
    errorCallback && errorCallback(e);
});
listeners.push(p);

timers[id] = {
    timer:window.setInterval(function() {
        if (accel) {
            successCallback(accel);
        }
    }, frequency),
    listeners:p
};

if (running) {
    // If we're already running then immediately invoke the success callback
    // but only if we have retrieved a value, sample code does not check for null ...
    if (accel) {
```

```
                successCallback(accel);
            }
        } else {
            start();
        }

        return id;
    },

    /**
     * Clears the specified accelerometer watch.
     *
     * @param {String} id                The id of the watch returned from
#watchAcceleration.
     */
    clearWatch: function(id) {
        // Stop javascript timer & remove from timer list
        if (id && timers[id]) {
            window.clearInterval(timers[id].timer);
            removeListeners(timers[id].listeners);
```

```
      delete timers[id];
    }
  }
};


module.exports = accelerometer;


});


// file: lib/common/plugin/accelerometer/symbols.js
define("cordova/plugin/accelerometer/symbols", function(require, exports, module) {



var modulemapper = require('cordova/modulemapper');


modulemapper.defaults('cordova/plugin/Acceleration', 'Acceleration');
modulemapper.defaults('cordova/plugin/accelerometer', 'navigator.accelerometer');
```

```
});

// file: lib/android/plugin/android/app.js
define("cordova/plugin/android/app", function(require, exports, module) {

var exec = require('cordova/exec');

module.exports = {
  /**
   * Clear the resource cache.
   */
  clearCache:function() {
    exec(null, null, "App", "clearCache", []);
  },

  /**
   * Load the url into the webview or into new browser instance.
   *
   * @param url           The URL to load
   * @param props         Properties that can be passed in to the activity:
   *      wait: int                          => wait msec before loading URL
   *      loadingDialog: "Title,Message"      => display a native loading dialog
   *      loadUrlTimeoutValue: int            => time in msec to wait before triggering a timeout error
   *      clearHistory: boolean              => clear webview history (default=false)
   *      openExternal: boolean              => open in a new browser (default=false)
   *
   * Example:
   *             navigator.app.loadUrl("http://server/myapp/index.html",   {wait:2000, loadingDialog:"Wait,Loading App", loadUrlTimeoutValue: 60000});
   */
  loadUrl:function(url, props) {
    exec(null, null, "App", "loadUrl", [url, props]);
  },


  /**
   * Cancel loadUrl that is waiting to be loaded.
   */
  cancelLoadUrl:function() {
    exec(null, null, "App", "cancelLoadUrl", []);
  },

  /**
   * Clear web history in this web view.
   * Instead of BACK button loading the previous web page, it will exit the app.
   */
  clearHistory:function() {
    exec(null, null, "App", "clearHistory", []);
  },

  /**
   * Go to previous page displayed.
   * This is the same as pressing the backbutton on Android device.
   */
  backHistory:function() {
    exec(null, null, "App", "backHistory", []);
  },
  /**
   * Override the default behavior of the Android back button.
   * If overridden, when the back button is pressed, the "backKeyDown" JavaScript event will be fired.
   *
   * Note: The user should not have to call this method.  Instead, when the user
   *       registers for the "backbutton" event, this is automatically done.
   *
   * @param override        T=override, F=cancel override
   */
  overrideBackbutton:function(override) {
    exec(null, null, "App", "overrideBackbutton", [override]);
  },

  /**
   * Exit and terminate the application.
   */
  exitApp:function() {
    return exec(null, null, "App", "exitApp", []);
  }
```

```
};


});


// file: lib/android/plugin/android/device.js

define("cordova/plugin/android/device", function(require, exports, module) {


var channel = require('cordova/channel'),

    utils = require('cordova/utils'),

    exec = require('cordova/exec'),

    app = require('cordova/plugin/android/app');


module.exports = {

    /*

     * DEPRECATED

     * This is only for Android.

     *

     * You must explicitly override the back button.

     */
```

```
  overrideBackButton:function() {
    console.log("Device.overrideBackButton() is deprecated. Use App.overrideBackbutton(true).");
    app.overrideBackbutton(true);
  },

  /*
   * DEPRECATED
   * This is only for Android.
   *
   * This resets the back button to the default behavior
   */
  resetBackButton:function() {
    console.log("Device.resetBackButton() is deprecated. Use App.overrideBackbutton(false).");
    app.overrideBackbutton(false);
  },

  /*
   * DEPRECATED
```

```
     * This is only for Android.
     *
     * This terminates the activity!
     */
    exitApp:function() {
        console.log("Device.exitApp() is deprecated.  Use App.exitApp().");
        app.exitApp();
    }
};


});


// file: lib/android/plugin/android/nativeapiprovider.js
define("cordova/plugin/android/nativeapiprovider", function(require, exports, module) {


/**
 * Exports the ExposedJsApi.java object if available, otherwise exports the PromptBasedNativeApi.
 */
```

```
var nativeApi = this._cordovaNative ||
require('cordova/plugin/android/promptbasednativeapi');
var currentApi = nativeApi;


module.exports = {
   get: function() { return currentApi; },
   setPreferPrompt: function(value) {
      currentApi = value ? require('cordova/plugin/android/promptbasednativeapi')
: nativeApi;
   },
   // Used only by tests.
   set: function(value) {
      currentApi = value;
   }
};


});


// file: lib/android/plugin/android/notification.js
```

```
define("cordova/plugin/android/notification", function(require, exports, module) {

var exec = require('cordova/exec');

/**
 * Provides Android enhanced notification API.
 */
module.exports = {
    activityStart : function(title, message) {
        // If title and message not specified then mimic Android behavior of
        // using default strings.
        if (typeof title === "undefined" && typeof message == "undefined") {
            title = "Busy";
            message = 'Please wait...';
        }

        exec(null, null, 'Notification', 'activityStart', [ title, message ]);
    },
/**
 * Close an activity dialog
 */
activityStop : function() {
   exec(null, null, 'Notification', 'activityStop', []);
},


/**
 * Display a progress dialog with progress bar that goes from 0 to 100.
 *
 * @param {String}
 *           title Title of the progress dialog.
 * @param {String}
 *           message Message to display in the dialog.
 */
progressStart : function(title, message) {
   exec(null, null, 'Notification', 'progressStart', [ title, message ]);
},
    /**
     * Close the progress dialog.
     */
    progressStop : function() {
        exec(null, null, 'Notification', 'progressStop', []);
    },

    /**
     * Set the progress dialog value.
     *
     * @param {Number}
     *            value 0-100
     */
    progressValue : function(value) {
        exec(null, null, 'Notification', 'progressValue', [ value ]);
    }
};

});
```

```
// file: lib/android/plugin/android/promptbasednativeapi.js
define("cordova/plugin/android/promptbasednativeapi", function(require, exports, module) {

/**
 * Implements the API of ExposedJsApi.java, but uses prompt() to communicate.
 * This is used only on the 2.3 simulator, where addJavascriptInterface() is broken.
 */

module.exports = {
    exec: function(service, action, callbackId, argsJson) {
        return prompt(argsJson, 'gap:'+JSON.stringify([service, action, callbackId]));
    },
    setNativeToJsBridgeMode: function(value) {
        prompt(value, 'gap_bridge_mode:');
    },
    retrieveJsMessages: function() {
        return prompt('', 'gap_poll:');
    }

};


});


// file: lib/android/plugin/android/storage.js

define("cordova/plugin/android/storage", function(require, exports, module) {


var utils = require('cordova/utils'),

    exec = require('cordova/exec'),

    channel = require('cordova/channel');


var queryQueue = {};


/**

 * SQL result set object

 * PRIVATE METHOD

 * @constructor

 */
```

```
var DroidDB_Rows = function() {
   this.resultSet = [];    // results array
   this.length = 0;        // number of rows
};

/**
 * Get item from SQL result set
 *
 * @param row           The row number to return
 * @return              The row object
 */
DroidDB_Rows.prototype.item = function(row) {
   return this.resultSet[row];
};

/**
 * SQL result set that is returned to user.
 * PRIVATE METHOD
 * @constructor
 */
var DroidDB_Result = function() {
    this.rows = new DroidDB_Rows();
};


/**
 * Callback from native code when query is complete.
 * PRIVATE METHOD
 *
 * @param id   Query id
 */
function completeQuery(id, data) {
    var query = queryQueue[id];
    if (query) {
        try {
            delete queryQueue[id];


            // Get transaction
            var tx = query.tx;
```

```
// If transaction hasn't failed
// Note: We ignore all query results if previous query
//      in the same transaction failed.
if (tx && tx.queryList[id]) {

    // Save query results
    var r = new DroidDB_Result();
    r.rows.resultSet = data;
    r.rows.length = data.length;
    try {
        if (typeof query.successCallback === 'function') {
            query.successCallback(query.tx, r);
        }
    } catch (ex) {
        console.log("executeSql error calling user success callback: "+ex);
    }

    tx.queryComplete(id);
```

```
            }
        } catch (e) {
            console.log("executeSql error: "+e);
        }
    }
}

/**
 * Callback from native code when query fails
 * PRIVATE METHOD
 *
 * @param reason        Error message
 * @param id           Query id
 */
function failQuery(reason, id) {
    var query = queryQueue[id];
    if (query) {
        try {
            delete queryQueue[id];
```

```
// Get transaction
var tx = query.tx;

// If transaction hasn't failed
// Note: We ignore all query results if previous query
//      in the same transaction failed.
if (tx && tx.queryList[id]) {
   tx.queryList = {};

   try {
      if (typeof query.errorCallback === 'function') {
         query.errorCallback(query.tx, reason);
      }
   } catch (ex) {
      console.log("executeSql error calling user error callback: "+ex);
   }

   tx.queryFailed(id, reason);
```

```
            }

        } catch (e) {
            console.log("executeSql error: "+e);
        }
    }
}

/**
 * SQL query object
 * PRIVATE METHOD
 *
 * @constructor
 * @param tx                The transaction object that this query belongs to
 */
var DroidDB_Query = function(tx) {

    // Set the id of the query
    this.id = utils.createUUID();
    // Add this query to the queue

    queryQueue[this.id] = this;


    // Init result

    this.resultSet = [];


    // Set transaction that this query belongs to

    this.tx = tx;


    // Add this query to transaction list

    this.tx.queryList[this.id] = this;


    // Callbacks

    this.successCallback = null;

    this.errorCallback = null;


};
```

```
/**
 * Transaction object
 * PRIVATE METHOD
 * @constructor
 */
var DroidDB_Tx = function() {

    // Set the id of the transaction
    this.id = utils.createUUID();

    // Callbacks
    this.successCallback = null;
    this.errorCallback = null;

    // Query list
    this.queryList = {};
};

/**
```

```
 * Mark query in transaction as complete.
 * If all queries are complete, call the user's transaction success callback.
 *
 * @param id                Query id
 */
DroidDB_Tx.prototype.queryComplete = function(id) {
   delete this.queryList[id];

   // If no more outstanding queries, then fire transaction success
   if (this.successCallback) {
      var count = 0;
      var i;
      for (i in this.queryList) {
         if (this.queryList.hasOwnProperty(i)) {
            count++;
         }
      }
      if (count === 0) {
         try {
                this.successCallback();
            } catch(e) {
                console.log("Transaction error calling user success callback: " + e);
            }
        }
    }
};

/**
 * Mark query in transaction as failed.
 *
 * @param id                Query id
 * @param reason            Error message
 */
DroidDB_Tx.prototype.queryFailed = function(id, reason) {

    // The sql queries in this transaction have already been run, since
    // we really don't have a real transaction implemented in native code.
    // However, the user callbacks for the remaining sql queries in transaction
    // will not be called.
    this.queryList = {};

    if (this.errorCallback) {
        try {
            this.errorCallback(reason);
        } catch(e) {
            console.log("Transaction error calling user error callback: " + e);
        }
    }
};

/**
 * Execute SQL statement
 *
 * @param sql                   SQL statement to execute
 * @param params                Statement parameters
 * @param successCallback       Success callback
 * @param errorCallback         Error callback
 */
DroidDB_Tx.prototype.executeSql = function(sql, params, successCallback, errorCallback) {

    // Init params array
    if (typeof params === 'undefined') {
        params = [];
    }

    // Create query and add to queue
    var query = new DroidDB_Query(this);
    queryQueue[query.id] = query;

    // Save callbacks
    query.successCallback = successCallback;
    query.errorCallback = errorCallback;

    // Call native code
    exec(null, null, "Storage", "executeSql", [sql, params, query.id]);
};
```

```
var DatabaseShell = function() {
};


/**
 * Start a transaction.
 * Does not support rollback in event of failure.
 *
 * @param process {Function}            The transaction function
 * @param successCallback {Function}
 * @param errorCallback {Function}
 */
DatabaseShell.prototype.transaction = function(process, errorCallback, successCallback) {
    var tx = new DroidDB_Tx();
    tx.successCallback = successCallback;
    tx.errorCallback = errorCallback;
    try {
        process(tx);
    } catch (e) {
        console.log("Transaction error: "+e);
        if (tx.errorCallback) {
            try {
                tx.errorCallback(e);
            } catch (ex) {
                console.log("Transaction error calling user error callback: "+e);
            }
        }
    }
};

/**
 * Open database
 *
 * @param name              Database name
 * @param version           Database version
 * @param display_name      Database display name
 * @param size              Database size in bytes
 * @return                  Database object
 */
var DroidDB_openDatabase = function(name, version, display_name, size) {
   exec(null, null, "Storage", "openDatabase", [name, version, display_name, size]);
   var db = new DatabaseShell();
   return db;
};



module.exports = {
  openDatabase:DroidDB_openDatabase,
  failQuery:failQuery,
  completeQuery:completeQuery
};


});


// file: lib/android/plugin/android/storage/openDatabase.js
define("cordova/plugin/android/storage/openDatabase", function(require, exports, module) {
```

```
var modulemapper = require('cordova/modulemapper'),
    storage = require('cordova/plugin/android/storage');

var originalOpenDatabase = modulemapper.getOriginalSymbol(window, 'openDatabase');

module.exports = function(name, version, desc, size) {
    // First patch WebSQL if necessary
    if (!originalOpenDatabase) {
        // Not defined, create an openDatabase function for all to use!
        return storage.openDatabase.apply(this, arguments);
    }

    // Defined, but some Android devices will throw a SECURITY_ERR -
    // so we wrap the whole thing in a try-catch and shim in our own
    // if the device has Android bug 16175.
    try {
        return originalOpenDatabase(name, version, desc, size);
    } catch (ex) {
        if (ex.code !== 18) {
            throw ex;
        }
    }
    return storage.openDatabase(name, version, desc, size);
};



});


// file: lib/android/plugin/android/storage/symbols.js
define("cordova/plugin/android/storage/symbols", function(require, exports, module) {


var modulemapper = require('cordova/modulemapper');
```

```
modulemapper.clobbers('cordova/plugin/android/storage/openDatabase',
'openDatabase');


});


// file: lib/common/plugin/battery.js
define("cordova/plugin/battery", function(require, exports, module) {

/**
 * This class contains information about the current battery status.
 * @constructor
 */
var cordova = require('cordova'),
    exec = require('cordova/exec');

function handlers() {
  return battery.channels.batterystatus.numHandlers +
         battery.channels.batterylow.numHandlers +
         battery.channels.batterycritical.numHandlers;
}

var Battery = function() {
    this._level = null;
    this._isPlugged = null;
    // Create new event handlers on the window (returns a channel instance)
    this.channels = {
     batterystatus:cordova.addWindowEventHandler("batterystatus"),
     batterylow:cordova.addWindowEventHandler("batterylow"),
     batterycritical:cordova.addWindowEventHandler("batterycritical")
    };
    for (var key in this.channels) {
      this.channels[key].onHasSubscribersChange = Battery.onHasSubscribersChange;
    }
};
/**
 * Event handlers for when callbacks get registered for the battery.
 * Keep track of how many handlers we have so we can start and stop the native battery listener
 * appropriately (and hopefully save on battery life!).
 */
Battery.onHasSubscribersChange = function() {
  // If we just registered the first handler, make sure native listener is started.
  if (this.numHandlers === 1 && handlers() === 1) {
      exec(battery._status, battery._error, "Battery", "start", []);
  } else if (handlers() === 0) {
      exec(null, null, "Battery", "stop", []);
  }
};

/**
 * Callback for battery status
 *
 * @param {Object} info            keys: level, isPlugged
 */
Battery.prototype._status = function(info) {
    if (info) {
        var me = battery;
    var level = info.level;
        if (me._level !== level || me._isPlugged !== info.isPlugged) {
            // Fire batterystatus event
            cordova.fireWindowEvent("batterystatus", info);

            // Fire low battery event
            if (level === 20 || level === 5) {
                if (level === 20) {
                    cordova.fireWindowEvent("batterylow", info);
                }
                else {
                    cordova.fireWindowEvent("batterycritical", info);
                }
            }
        }
        me._level = level;
        me._isPlugged = info.isPlugged;
    }
};
```

```
/**
 * Error callback for battery start
 */
Battery.prototype._error = function(e) {
    console.log("Error initializing Battery: " + e);
};

var battery = new Battery();

module.exports = battery;


});


// file: lib/common/plugin/battery/symbols.js
define("cordova/plugin/battery/symbols", function(require, exports, module) {



var modulemapper = require('cordova/modulemapper');
```

```
modulemapper.defaults('cordova/plugin/battery', 'navigator.battery');

});

// file: lib/common/plugin/camera/symbols.js
define("cordova/plugin/camera/symbols", function(require, exports, module) {


var modulemapper = require('cordova/modulemapper');

modulemapper.defaults('cordova/plugin/Camera', 'navigator.camera');
modulemapper.defaults('cordova/plugin/CameraConstants', 'Camera');
modulemapper.defaults('cordova/plugin/CameraPopoverOptions',
'CameraPopoverOptions');

});

// file: lib/common/plugin/capture.js
define("cordova/plugin/capture", function(require, exports, module) {
```

```
var exec = require('cordova/exec'),
    MediaFile = require('cordova/plugin/MediaFile');

/**
 * Launches a capture of different types.
 *
 * @param (DOMString} type
 * @param {Function} successCB
 * @param {Function} errorCB
 * @param {CaptureVideoOptions} options
 */
function _capture(type, successCallback, errorCallback, options) {
    var win = function(pluginResult) {
        var mediaFiles = [];
        var i;
        for (i = 0; i < pluginResult.length; i++) {
            var mediaFile = new MediaFile();
            mediaFile.name = pluginResult[i].name;
            mediaFile.fullPath = pluginResult[i].fullPath;
            mediaFile.type = pluginResult[i].type;
            mediaFile.lastModifiedDate = pluginResult[i].lastModifiedDate;
            mediaFile.size = pluginResult[i].size;
            mediaFiles.push(mediaFile);
        }
        successCallback(mediaFiles);
    };
    exec(win, errorCallback, "Capture", type, [options]);
}
/**
 * The Capture interface exposes an interface to the camera and microphone of the hosting device.
 */
function Capture() {
    this.supportedAudioModes = [];
    this.supportedImageModes = [];
    this.supportedVideoModes = [];
}
```

```
/**
 * Launch audio recorder application for recording audio clip(s).
 *
 * @param {Function} successCB
 * @param {Function} errorCB
 * @param {CaptureAudioOptions} options
 */
Capture.prototype.captureAudio = function(successCallback, errorCallback, options){
   _capture("captureAudio", successCallback, errorCallback, options);
};

/**
 * Launch camera application for taking image(s).
 *
 * @param {Function} successCB
 * @param {Function} errorCB
 * @param {CaptureImageOptions} options
 */
```

```
Capture.prototype.captureImage = function(successCallback, errorCallback, options){
   _capture("captureImage", successCallback, errorCallback, options);
};

/**
 * Launch device camera application for recording video(s).
 *
 * @param {Function} successCB
 * @param {Function} errorCB
 * @param {CaptureVideoOptions} options
 */
Capture.prototype.captureVideo = function(successCallback, errorCallback, options){
   _capture("captureVideo", successCallback, errorCallback, options);
};


module.exports = new Capture();
```

```
});


// file: lib/common/plugin/capture/symbols.js

define("cordova/plugin/capture/symbols", function(require, exports, module) {


var modulemapper = require('cordova/modulemapper');


modulemapper.clobbers('cordova/plugin/CaptureError', 'CaptureError');

modulemapper.clobbers('cordova/plugin/CaptureAudioOptions',
'CaptureAudioOptions');

modulemapper.clobbers('cordova/plugin/CaptureImageOptions',
'CaptureImageOptions');

modulemapper.clobbers('cordova/plugin/CaptureVideoOptions',
'CaptureVideoOptions');

modulemapper.clobbers('cordova/plugin/ConfigurationData',
'ConfigurationData');

modulemapper.clobbers('cordova/plugin/MediaFile', 'MediaFile');

modulemapper.clobbers('cordova/plugin/MediaFileData', 'MediaFileData');

modulemapper.clobbers('cordova/plugin/capture', 'navigator.device.capture');


});
```

```
// file: lib/common/plugin/compass.js

define("cordova/plugin/compass", function(require, exports, module) {


var argscheck = require('cordova/argscheck'),

    exec = require('cordova/exec'),

    utils = require('cordova/utils'),

    CompassHeading = require('cordova/plugin/CompassHeading'),

    CompassError = require('cordova/plugin/CompassError'),

    timers = {},

    compass = {

        /**

         * Asynchronously acquires the current heading.

         * @param {Function} successCallback The function to call when the
heading

         * data is available

         * @param {Function} errorCallback The function to call when there is an
error

         * getting the heading data.
         * @param {CompassOptions} options The options for getting the heading
data (not used).
         */
        getCurrentHeading:function(successCallback, errorCallback, options) {
            argscheck.checkArgs('fFO', 'compass.getCurrentHeading', arguments);

            var win = function(result) {
                var ch = new CompassHeading(result.magneticHeading,
result.trueHeading, result.headingAccuracy, result.timestamp);
                successCallback(ch);
            };
            var fail = errorCallback && function(code) {
                var ce = new CompassError(code);
                errorCallback(ce);
            };

            // Get heading
            exec(win, fail, "Compass", "getHeading", [options]);
        },
        /**
         * Asynchronously acquires the heading repeatedly at a given interval.
         * @param {Function} successCallback The function to call each time the
heading
         * data is available
         * @param {Function} errorCallback The function to call when there is an
error
         * getting the heading data.
         * @param {HeadingOptions} options The options for getting the heading
data
         * such as timeout and the frequency of the watch. For iOS, filter parameter
         * specifies to watch via a distance filter rather than time.
         */
        watchHeading:function(successCallback, errorCallback, options) {
            argscheck.checkArgs('fFO', 'compass.watchHeading', arguments);
            // Default interval (100 msec)
            var frequency = (options !== undefined && options.frequency !==
undefined) ? options.frequency : 100;
            var filter = (options !== undefined && options.filter !== undefined) ?
options.filter : 0;
```

```
        var id = utils.createUUID();

        if (filter > 0) {

            // is an iOS request for watch by filter, no timer needed

            timers[id] = "iOS";

            compass.getCurrentHeading(successCallback, errorCallback, options);

        } else {

            // Start watch timer to get headings

            timers[id] = window.setInterval(function() {

                compass.getCurrentHeading(successCallback, errorCallback);

            }, frequency);

        }



        return id;

    },



    /**

     * Clears the specified heading watch.

     * @param {String} watchId The ID of the watch returned from #watchHeading.

     */
```

```
        clearWatch:function(id) {
            // Stop javascript timer & remove from timer list
            if (id && timers[id]) {
                if (timers[id] != "iOS") {
                    clearInterval(timers[id]);
                } else {
                    // is iOS watch by filter so call into device to stop
                    exec(null, null, "Compass", "stopHeading", []);
                }
                delete timers[id];
            }
        }
    };

module.exports = compass;

});

// file: lib/common/plugin/compass/symbols.js
```

```
define("cordova/plugin/compass/symbols", function(require, exports, module) {


var modulemapper = require('cordova/modulemapper');

modulemapper.clobbers('cordova/plugin/CompassHeading', 'CompassHeading');
modulemapper.clobbers('cordova/plugin/CompassError', 'CompassError');
modulemapper.clobbers('cordova/plugin/compass', 'navigator.compass');

});

// file: lib/common/plugin/console-via-logger.js
define("cordova/plugin/console-via-logger", function(require, exports, module) {

//------------------------------------------------------------------------------

var logger = require("cordova/plugin/logger");
var utils  = require("cordova/utils");
```

```
//----------------------------------------------------------------------------
// object that we're exporting
//----------------------------------------------------------------------------
var console = module.exports;

//----------------------------------------------------------------------------
// copy of the original console object
//----------------------------------------------------------------------------
var WinConsole = window.console;

//----------------------------------------------------------------------------
// whether to use the logger
//----------------------------------------------------------------------------
var UseLogger = false;

//----------------------------------------------------------------------------
// Timers
//----------------------------------------------------------------------------
var Timers = {};
```

```
//----------------------------------------------------------------------------
// used for unimplemented methods
//----------------------------------------------------------------------------
function noop() {}

//----------------------------------------------------------------------------
// used for unimplemented methods
//----------------------------------------------------------------------------
console.useLogger = function (value) {
    if (arguments.length) UseLogger = !!value;

    if (UseLogger) {
        if (logger.useConsole()) {
            throw new Error("console and logger are too intertwingly");
        }
    }

    return UseLogger;
};


//--------------------------------------------------------------------------
console.log = function() {
    if (logger.useConsole()) return;
    logger.log.apply(logger, [].slice.call(arguments));
};


//--------------------------------------------------------------------------
console.error = function() {
    if (logger.useConsole()) return;
    logger.error.apply(logger, [].slice.call(arguments));
};


//--------------------------------------------------------------------------
console.warn = function() {
    if (logger.useConsole()) return;
    logger.warn.apply(logger, [].slice.call(arguments));
};
```

```
//------------------------------------------------------------------------------
console.info = function() {
    if (logger.useConsole()) return;
    logger.info.apply(logger, [].slice.call(arguments));
};

//------------------------------------------------------------------------------
console.debug = function() {
    if (logger.useConsole()) return;
    logger.debug.apply(logger, [].slice.call(arguments));
};

//------------------------------------------------------------------------------
console.assert = function(expression) {
    if (expression) return;

    var message = logger.format.apply(logger.format, [].slice.call(arguments, 1));
    console.log("ASSERT: " + message);
};

//---------------------------------------------------------------------------

console.clear = function() {};

//---------------------------------------------------------------------------

console.dir = function(object) {

   console.log("%o", object);

};

//---------------------------------------------------------------------------

console.dirxml = function(node) {

   console.log(node.innerHTML);

};

//---------------------------------------------------------------------------

console.trace = noop;

//---------------------------------------------------------------------------
```

```
console.group = console.log;


//----------------------------------------------------------------------------

console.groupCollapsed = console.log;


//----------------------------------------------------------------------------

console.groupEnd = noop;


//----------------------------------------------------------------------------

console.time = function(name) {

   Timers[name] = new Date().valueOf();

};


//----------------------------------------------------------------------------

console.timeEnd = function(name) {

   var timeStart = Timers[name];

   if (!timeStart) {

      console.warn("unknown timer: " + name);

      return;
```

```
    }

    var timeElapsed = new Date().valueOf() - timeStart;
    console.log(name + ": " + timeElapsed + "ms");
};

//---------------------------------------------------------------------------
console.timeStamp = noop;

//---------------------------------------------------------------------------
console.profile = noop;

//---------------------------------------------------------------------------
console.profileEnd = noop;

//---------------------------------------------------------------------------
console.count = noop;

//---------------------------------------------------------------------------
```

```
console.exception = console.log;

//-----------------------------------------------------------------------------
console.table = function(data, columns) {
    console.log("%o", data);
};

//-----------------------------------------------------------------------------
// return a new function that calls both functions passed as args
//-----------------------------------------------------------------------------
function wrappedOrigCall(orgFunc, newFunc) {
    return function() {
        var args = [].slice.call(arguments);
        try { orgFunc.apply(WinConsole, args); } catch (e) {}
        try { newFunc.apply(console,    args); } catch (e) {}
    };
}

//-----------------------------------------------------------------------------
```

```
// For every function that exists in the original console object, that
// also exists in the new console object, wrap the new console method
// with one that calls both
//----------------------------------------------------------------------------
for (var key in console) {
    if (typeof WinConsole[key] == "function") {
        console[key] = wrappedOrigCall(WinConsole[key], console[key]);
    }
}


});


// file: lib/common/plugin/contacts.js
define("cordova/plugin/contacts", function(require, exports, module) {


var argscheck = require('cordova/argscheck'),
    exec = require('cordova/exec'),
    ContactError = require('cordova/plugin/ContactError'),
    utils = require('cordova/utils'),
    Contact = require('cordova/plugin/Contact');

/**
* Represents a group of Contacts.
* @constructor
*/
var contacts = {
    /**
     * Returns an array of Contacts matching the search criteria.
     * @param fields that should be searched
     * @param successCB success callback
     * @param errorCB error callback
     * @param {ContactFindOptions} options that can be applied to contact searching
     * @return array of Contacts matching search criteria
     */
    find:function(fields, successCB, errorCB, options) {
        argscheck.checkArgs('afFO', 'contacts.find', arguments);
        if (!fields.length) {
```

```
      errorCB && errorCB(new ContactError(ContactError.INVALID_ARGUMENT_ERROR));
    } else {
      var win = function(result) {
        var cs = [];
        for (var i = 0, l = result.length; i < l; i++) {
          cs.push(contacts.create(result[i]));
        }
        successCB(cs);
      };
      exec(win, errorCB, "Contacts", "search", [fields, options]);
    }
  },

  /**
   * This function creates a new contact, but it does not persist the contact
   * to device storage. To persist the contact to device storage, invoke
   * contact.save().
   * @param properties an object whose properties will be examined to create a new Contact
```

```
     * @returns new Contact object
     */
    create:function(properties) {
        argscheck.checkArgs('O', 'contacts.create', arguments);
        var contact = new Contact();
        for (var i in properties) {
            if (typeof contact[i] !== 'undefined' && properties.hasOwnProperty(i)) {
                contact[i] = properties[i];
            }
        }
        return contact;
    }
};

module.exports = contacts;


});


// file: lib/common/plugin/contacts/symbols.js
```

```
define("cordova/plugin/contacts/symbols", function(require, exports, module) {



var modulemapper = require('cordova/modulemapper');


modulemapper.clobbers('cordova/plugin/contacts', 'navigator.contacts');
modulemapper.clobbers('cordova/plugin/Contact', 'Contact');
modulemapper.clobbers('cordova/plugin/ContactAddress', 'ContactAddress');
modulemapper.clobbers('cordova/plugin/ContactError', 'ContactError');
modulemapper.clobbers('cordova/plugin/ContactField', 'ContactField');
modulemapper.clobbers('cordova/plugin/ContactFindOptions',
'ContactFindOptions');
modulemapper.clobbers('cordova/plugin/ContactName', 'ContactName');
modulemapper.clobbers('cordova/plugin/ContactOrganization',
'ContactOrganization');


});


// file: lib/common/plugin/device.js
define("cordova/plugin/device", function(require, exports, module) {
```

```
var argscheck = require('cordova/argscheck'),
    channel = require('cordova/channel'),
    utils = require('cordova/utils'),
    exec = require('cordova/exec');

// Tell cordova channel to wait on the CordovaInfoReady event
channel.waitForInitialization('onCordovaInfoReady');

/**
 * This represents the mobile device, and provides properties for inspecting the model, version, UUID of the
 * phone, etc.
 * @constructor
 */
function Device() {
    this.available = false;
    this.platform = null;
    this.version = null;
    this.uuid = null;
this.cordova = null;
this.model = null;

var me = this;

channel.onCordovaReady.subscribe(function() {
    me.getInfo(function(info) {
        var buildLabel = info.cordova;
        if (buildLabel != CORDOVA_JS_BUILD_LABEL) {
            buildLabel += ' JS=' + CORDOVA_JS_BUILD_LABEL;
        }
        me.available = true;
        me.platform = info.platform;
        me.version = info.version;
        me.uuid = info.uuid;
        me.cordova = buildLabel;
        me.model = info.model;
        channel.onCordovaInfoReady.fire();
    },function(e) {
            me.available = false;

            utils.alert("[ERROR] Error initializing Cordova: " + e);

        });

    });

}


/**

 * Get device info

 *

 * @param {Function} successCallback The function to call when the heading data is available

 * @param {Function} errorCallback The function to call when there is an error getting the heading data. (OPTIONAL)

 */

Device.prototype.getInfo = function(successCallback, errorCallback) {

    argscheck.checkArgs('fF', 'Device.getInfo', arguments);

    exec(successCallback, errorCallback, "Device", "getDeviceInfo", []);

};


module.exports = new Device();
```

```
});

// file: lib/android/plugin/device/symbols.js
define("cordova/plugin/device/symbols", function(require, exports, module) {



var modulemapper = require('cordova/modulemapper');


modulemapper.clobbers('cordova/plugin/device', 'device');
modulemapper.merges('cordova/plugin/android/device', 'device');


});


// file: lib/common/plugin/echo.js
define("cordova/plugin/echo", function(require, exports, module) {


var exec = require('cordova/exec'),
    utils = require('cordova/utils');
```

```
/**
 * Sends the given message through exec() to the Echo plugin, which sends it
back to the successCallback.
 * @param successCallback  invoked with a FileSystem object
 * @param errorCallback  invoked if error occurs retrieving file system
 * @param message  The string to be echoed.
 * @param forceAsync  Whether to force an async return value (for testing
native->js bridge).
 */
module.exports = function(successCallback, errorCallback, message, forceAsync)
{
    var action = 'echo';
    var messageIsMultipart = (utils.typeName(message) == "Array");
    var args = messageIsMultipart ? message : [message];

    if (utils.typeName(message) == 'ArrayBuffer') {
        if (forceAsync) {
            console.warn('Cannot echo ArrayBuffer with forced async, falling back to
sync.');
        }
        action += 'ArrayBuffer';
    } else if (messageIsMultipart) {
        if (forceAsync) {
            console.warn('Cannot echo MultiPart Array with forced async, falling back to sync.');
        }
        action += 'MultiPart';
    } else if (forceAsync) {
        action += 'Async';
    }

    exec(successCallback, errorCallback, "Echo", action, args);
};



});


// file: lib/android/plugin/file/symbols.js
define("cordova/plugin/file/symbols", function(require, exports, module) {
```

```
var modulemapper = require('cordova/modulemapper'),
    symbolshelper = require('cordova/plugin/file/symbolshelper');

symbolshelper(modulemapper.clobbers);

});

// file: lib/common/plugin/file/symbolshelper.js
define("cordova/plugin/file/symbolshelper", function(require, exports, module) {

module.exports = function(exportFunc) {
    exportFunc('cordova/plugin/DirectoryEntry', 'DirectoryEntry');
    exportFunc('cordova/plugin/DirectoryReader', 'DirectoryReader');
    exportFunc('cordova/plugin/Entry', 'Entry');
    exportFunc('cordova/plugin/File', 'File');
    exportFunc('cordova/plugin/FileEntry', 'FileEntry');
    exportFunc('cordova/plugin/FileError', 'FileError');
    exportFunc('cordova/plugin/FileReader', 'FileReader');
    exportFunc('cordova/plugin/FileSystem', 'FileSystem');

    exportFunc('cordova/plugin/FileUploadOptions', 'FileUploadOptions');

    exportFunc('cordova/plugin/FileUploadResult', 'FileUploadResult');

    exportFunc('cordova/plugin/FileWriter', 'FileWriter');

    exportFunc('cordova/plugin/Flags', 'Flags');

    exportFunc('cordova/plugin/LocalFileSystem', 'LocalFileSystem');

    exportFunc('cordova/plugin/Metadata', 'Metadata');

    exportFunc('cordova/plugin/ProgressEvent', 'ProgressEvent');

    exportFunc('cordova/plugin/requestFileSystem', 'requestFileSystem');

    exportFunc('cordova/plugin/resolveLocalFileSystemURI',
'resolveLocalFileSystemURI');

};


});


// file: lib/common/plugin/filetransfer/symbols.js

define("cordova/plugin/filetransfer/symbols", function(require, exports, module)
{
```

```
var modulemapper = require('cordova/modulemapper');

modulemapper.clobbers('cordova/plugin/FileTransfer', 'FileTransfer');
modulemapper.clobbers('cordova/plugin/FileTransferError', 'FileTransferError');

});

// file: lib/common/plugin/geolocation.js
define("cordova/plugin/geolocation", function(require, exports, module) {

var argscheck = require('cordova/argscheck'),
    utils = require('cordova/utils'),
    exec = require('cordova/exec'),
    PositionError = require('cordova/plugin/PositionError'),
    Position = require('cordova/plugin/Position');

var timers = {};   // list of timers in use

// Returns default params, overrides if provided with values
```

```
function parseParameters(options) {
    var opt = {
        maximumAge: 0,
        enableHighAccuracy: false,
        timeout: Infinity
    };

    if (options) {
        if (options.maximumAge !== undefined && !isNaN(options.maximumAge) && options.maximumAge > 0) {
            opt.maximumAge = options.maximumAge;
        }
        if (options.enableHighAccuracy !== undefined) {
            opt.enableHighAccuracy = options.enableHighAccuracy;
        }
        if (options.timeout !== undefined && !isNaN(options.timeout)) {
            if (options.timeout < 0) {
                opt.timeout = 0;
            } else {
                opt.timeout = options.timeout;
            }
        }
    }

    return opt;
}

// Returns a timeout failure, closed over a specified timeout value and error callback.
function createTimeout(errorCallback, timeout) {
    var t = setTimeout(function() {
        clearTimeout(t);
        t = null;
        errorCallback({
            code:PositionError.TIMEOUT,
            message:"Position retrieval timed out."
        });
    }, timeout);
    return t;
}
```

```
var geolocation = {
   lastPosition:null, // reference to last known (cached) position returned
   /**
  * Asynchronously acquires the current position.
  *
  * @param {Function} successCallback    The function to call when the position data is available
  * @param {Function} errorCallback      The function to call when there is an error getting the heading position. (OPTIONAL)
  * @param {PositionOptions} options     The options for getting the position data. (OPTIONAL)
  */
   getCurrentPosition:function(successCallback, errorCallback, options) {
      argscheck.checkArgs('fFO', 'geolocation.getCurrentPosition', arguments);
      options = parseParameters(options);

      // Timer var that will fire an error callback if no position is retrieved from native
      // before the "timeout" param provided expires
      var timeoutTimer = {timer:null};
var win = function(p) {
    clearTimeout(timeoutTimer.timer);
    if (!(timeoutTimer.timer)) {
        // Timeout already happened, or native fired error callback for
        // this geo request.
        // Don't continue with success callback.
        return;
    }
    var pos = new Position(
        {
            latitude:p.latitude,
            longitude:p.longitude,
            altitude:p.altitude,
            accuracy:p.accuracy,
            heading:p.heading,
            velocity:p.velocity,
            altitudeAccuracy:p.altitudeAccuracy
        },
        (p.timestamp === undefined ? new Date() : ((p.timestamp instanceof Date) ? p.timestamp : new Date(p.timestamp)))
      );
      geolocation.lastPosition = pos;
      successCallback(pos);
    };
    var fail = function(e) {
      clearTimeout(timeoutTimer.timer);
      timeoutTimer.timer = null;
      var err = new PositionError(e.code, e.message);
      if (errorCallback) {
        errorCallback(err);
      }
    };

    // Check our cached position, if its timestamp difference with current time is less than the maximumAge, then just
    // fire the success callback with the cached position.
    if (geolocation.lastPosition && options.maximumAge && (((new Date()).getTime() - geolocation.lastPosition.timestamp.getTime()) <= options.maximumAge)) {
            successCallback(geolocation.lastPosition);
        // If the cached position check failed and the timeout was set to 0, error out with a TIMEOUT error object.
        } else if (options.timeout === 0) {
            fail({
                code:PositionError.TIMEOUT,
                message:"timeout value in PositionOptions set to 0 and no cached Position object available, or cached Position object's age exceeds provided PositionOptions' maximumAge parameter."
            });
        // Otherwise we have to call into native to retrieve a position.
        } else {
            if (options.timeout !== Infinity) {
                // If the timeout value was not set to Infinity (default), then
                // set up a timeout function that will fire the error callback
                // if no successful position was retrieved before timeout expired.
                timeoutTimer.timer = createTimeout(fail, options.timeout);
            } else {
                // This is here so the check in the win function doesn't mess stuff up
                // may seem weird but this guarantees timeoutTimer is
            // always truthy before we call into native
            timeoutTimer.timer = true;
        }
        exec(win,          fail,          "Geolocation",          "getLocation",
[options.enableHighAccuracy, options.maximumAge]);
      }
      return timeoutTimer;
  },
  /**
   * Asynchronously watches the geolocation for changes to geolocation.  When
a change occurs,
   * the successCallback is called with the new location.
   *
   * @param {Function} successCallback    The function to call each time the
location data is available
   * @param {Function} errorCallback      The function to call when there is an
error getting the location data. (OPTIONAL)
   * @param {PositionOptions} options     The options for getting the location
data such as frequency. (OPTIONAL)
   * @return String                       The watch id that must be passed to
#clearWatch to stop watching.
```

```
     */
    watchPosition:function(successCallback, errorCallback, options) {
        argscheck.checkArgs('fFO', 'geolocation.getCurrentPosition', arguments);
        options = parseParameters(options);

        var id = utils.createUUID();

        // Tell device to get a position ASAP, and also retrieve a reference to the timeout timer generated in getCurrentPosition
        timers[id] = geolocation.getCurrentPosition(successCallback, errorCallback, options);

        var fail = function(e) {
            clearTimeout(timers[id].timer);
            var err = new PositionError(e.code, e.message);
            if (errorCallback) {
                errorCallback(err);
            }
        };
    var win = function(p) {
      clearTimeout(timers[id].timer);
      if (options.timeout !== Infinity) {
        timers[id].timer = createTimeout(fail, options.timeout);
      }
      var pos = new Position(
        {
          latitude:p.latitude,
          longitude:p.longitude,
          altitude:p.altitude,
          accuracy:p.accuracy,
          heading:p.heading,
          velocity:p.velocity,
          altitudeAccuracy:p.altitudeAccuracy
        },
        (p.timestamp === undefined ? new Date() : ((p.timestamp instanceof Date) ? p.timestamp : new Date(p.timestamp)))
      );
      geolocation.lastPosition = pos;
      successCallback(pos);
```

```
        };

        exec(win, fail, "Geolocation", "addWatch", [id, options.enableHighAccuracy]);

        return id;
    },
    /**
     * Clears the specified heading watch.
     *
     * @param {String} id       The ID of the watch returned from #watchPosition
     */
    clearWatch:function(id) {
        if (id && timers[id] !== undefined) {
            clearTimeout(timers[id].timer);
            timers[id].timer = false;
            exec(null, null, "Geolocation", "clearWatch", [id]);
        }
    }
};
```

```
module.exports = geolocation;

});

// file: lib/common/plugin/geolocation/symbols.js
define("cordova/plugin/geolocation/symbols", function(require, exports, module) {


var modulemapper = require('cordova/modulemapper');

modulemapper.defaults('cordova/plugin/geolocation', 'navigator.geolocation');
modulemapper.clobbers('cordova/plugin/PositionError', 'PositionError');
modulemapper.clobbers('cordova/plugin/Position', 'Position');
modulemapper.clobbers('cordova/plugin/Coordinates', 'Coordinates');

});

// file: lib/common/plugin/globalization.js
```

```
define("cordova/plugin/globalization", function(require, exports, module) {

var argscheck = require('cordova/argscheck'),
    exec = require('cordova/exec'),
    GlobalizationError = require('cordova/plugin/GlobalizationError');

var globalization = {

/**
* Returns the string identifier for the client's current language.
* It returns the language identifier string to the successCB callback with a
* properties object as a parameter. If there is an error getting the language,
* then the errorCB callback is invoked.
*
* @param {Function} successCB
* @param {Function} errorCB
*
* @return Object.value {String}: The language identifier
*
```

```
* @error GlobalizationError.UNKNOWN_ERROR
*
* Example
*    globalization.getPreferredLanguage(function (language) {alert('language:' +
language.value + '\n');},
*                                function () {});
*/
getPreferredLanguage:function(successCB, failureCB) {
   argscheck.checkArgs('fF', 'Globalization.getPreferredLanguage', arguments);
   exec(successCB, failureCB, "Globalization","getPreferredLanguage", []);
},


/**
* Returns the string identifier for the client's current locale setting.
* It returns the locale identifier string to the successCB callback with a
* properties object as a parameter. If there is an error getting the locale,
* then the errorCB callback is invoked.
*
* @param {Function} successCB
* @param {Function} errorCB
```

```
*
* @return Object.value {String}: The locale identifier
*
* @error GlobalizationError.UNKNOWN_ERROR
*
* Example
*    globalization.getLocaleName(function (locale) {alert('locale:' + locale.value +
'\n');},
*                                function () {});
*/
getLocaleName:function(successCB, failureCB) {
   argscheck.checkArgs('fF', 'Globalization.getLocaleName', arguments);
   exec(successCB, failureCB, "Globalization","getLocaleName", []);
},


/**
* Returns a date formatted as a string according to the client's user preferences
and
```

```
* calendar using the time zone of the client. It returns the formatted date string to the
* successCB callback with a properties object as a parameter. If there is an error
* formatting the date, then the errorCB callback is invoked.
*
* The defaults are: formatLenght="short" and selector="date and time"
*
* @param {Date} date
* @param {Function} successCB
* @param {Function} errorCB
* @param {Object} options {optional}
*           formatLength {String}: 'short', 'medium', 'long', or 'full'
*           selector {String}: 'date', 'time', or 'date and time'
*
* @return Object.value {String}: The localized date string
*
* @error GlobalizationError.FORMATTING_ERROR
*
* Example
*    globalization.dateToString(new Date(),
*            function (date) {alert('date:' + date.value + '\n');},
*            function (errorCode) {alert(errorCode);},
*            {formatLength:'short'});
*/
dateToString:function(date, successCB, failureCB, options) {
   argscheck.checkArgs('dfFO', 'Globalization.dateToString', arguments);
   var dateValue = date.valueOf();
   exec(successCB, failureCB, "Globalization", "dateToString", [{"date":
dateValue, "options": options}]);
},


/**
* Parses a date formatted as a string according to the client's user
* preferences and calendar using the time zone of the client and returns
* the corresponding date object. It returns the date to the successCB
* callback with a properties object as a parameter. If there is an error
* parsing the date string, then the errorCB callback is invoked.
*
* The defaults are: formatLength="short" and selector="date and time"
```

```
*
* @param {String} dateString
* @param {Function} successCB
* @param {Function} errorCB
* @param {Object} options {optional}
*         formatLength {String}: 'short', 'medium', 'long', or 'full'
*         selector {String}: 'date', 'time', or 'date and time'
*
* @return   Object.year {Number}: The four digit year
*         Object.month {Number}: The month from (0 - 11)
*         Object.day {Number}: The day from (1 - 31)
*         Object.hour {Number}: The hour from (0 - 23)
*         Object.minute {Number}: The minute from (0 - 59)
*         Object.second {Number}: The second from (0 - 59)
*         Object.millisecond {Number}: The milliseconds (from 0 - 999),
*                         not available on all platforms
*
* @error GlobalizationError.PARSING_ERROR
*
```

```
* Example
*    globalization.stringToDate('4/11/2011',
*              function (date) { alert('Month:' + date.month + '\n' +
*                 'Day:' + date.day + '\n' +
*                 'Year:' + date.year + '\n');},
*              function (errorCode) {alert(errorCode);},
*              {selector:'date'});
*/
stringToDate:function(dateString, successCB, failureCB, options) {
   argscheck.checkArgs('sfFO', 'Globalization.stringToDate', arguments);
   exec(successCB, failureCB, "Globalization", "stringToDate", [{"dateString": dateString, "options": options}]);
},


/**
* Returns a pattern string for formatting and parsing dates according to the client's
* user preferences. It returns the pattern to the successCB callback with a
* properties object as a parameter. If there is an error obtaining the pattern,
* then the errorCB callback is invoked.
```

```
*
* The defaults are: formatLength="short" and selector="date and time"
*
* @param {Function} successCB
* @param {Function} errorCB
* @param {Object} options {optional}
*          formatLength {String}: 'short', 'medium', 'long', or 'full'
*          selector {String}: 'date', 'time', or 'date and time'
*
* @return   Object.pattern {String}: The date and time pattern for formatting and
parsing dates.
*                              The patterns follow Unicode Technical Standard #35
*                              http://unicode.org/reports/tr35/tr35-4.html
*          Object.timezone {String}: The abbreviated name of the time zone on the
client
*          Object.utc_offset {Number}: The current difference in seconds between
the client's
*                                time zone and coordinated universal time.
*           Object.dst_offset {Number}: The current daylight saving time offset in
seconds
*                                between the client's non-daylight saving's time zone
*                               and the client's daylight saving's time zone.
*
* @error GlobalizationError.PATTERN_ERROR
*
* Example
*    globalization.getDatePattern(
*              function (date) {alert('pattern:' + date.pattern + '\n');},
*              function () {},
*              {formatLength:'short'});
*/
getDatePattern:function(successCB, failureCB, options) {
   argscheck.checkArgs('fFO', 'Globalization.getDatePattern', arguments);
   exec(successCB, failureCB, "Globalization", "getDatePattern", [{"options": options}]);
},



/**
* Returns an array of either the names of the months or days of the week
* according to the client's user preferences and calendar. It returns the array of
names to the
* successCB callback with a properties object as a parameter. If there is an error
obtaining the
* names, then the errorCB callback is invoked.
*
* The defaults are: type="wide" and item="months"
*
* @param {Function} successCB
* @param {Function} errorCB
* @param {Object} options {optional}
*         type {String}: 'narrow' or 'wide'
*         item {String}: 'months', or 'days'
*
* @return Object.value {Array{String}}: The array of names starting from either
*                              the first month in the year or the
*                              first day of the week.
* @error GlobalizationError.UNKNOWN_ERROR
*
* Example
*    globalization.getDateNames(function (names) {

*       for(var i = 0; i < names.value.length; i++) {

*           alert('Month:' + names.value[i] + '\n');}},

*       function () {});

*/

getDateNames:function(successCB, failureCB, options) {

   argscheck.checkArgs('fFO', 'Globalization.getDateNames', arguments);

   exec(successCB, failureCB, "Globalization", "getDateNames", [{"options":
options}]);

},


/**

* Returns whether daylight savings time is in effect for a given date using the
client's

* time zone and calendar. It returns whether or not daylight savings time is in
effect

* to the successCB callback with a properties object as a parameter. If there is an
error

* reading the date, then the errorCB callback is invoked.

*

* @param {Date} date
* @param {Function} successCB

* @param {Function} errorCB

*

* @return Object.dst {Boolean}: The value "true" indicates that daylight savings
time is

*                          in effect for the given date and "false" indicate that it is not.

*

* @error GlobalizationError.UNKNOWN_ERROR

*

* Example

*    globalization.isDayLightSavingsTime(new Date(),

*             function (date) {alert('dst:' + date.dst + '\n');}

*             function () {});

*/

isDayLightSavingsTime:function(date, successCB, failureCB) {

   argscheck.checkArgs('dfF', 'Globalization.isDayLightSavingsTime',
arguments);

   var dateValue = date.valueOf();

   exec(successCB, failureCB, "Globalization", "isDayLightSavingsTime",
[{"date": dateValue}]);
```

```
},

/**
* Returns the first day of the week according to the client's user preferences and calendar.
* The days of the week are numbered starting from 1 where 1 is considered to be Sunday.
* It returns the day to the successCB callback with a properties object as a parameter.
* If there is an error obtaining the pattern, then the errorCB callback is invoked.
*
* @param {Function} successCB
* @param {Function} errorCB
*
* @return Object.value {Number}: The number of the first day of the week.
*
* @error GlobalizationError.UNKNOWN_ERROR
*
* Example
*    globalization.getFirstDayOfWeek(function (day)
*              { alert('Day:' + day.value + '\n');},
*              function () {});
*/
getFirstDayOfWeek:function(successCB, failureCB) {
   argscheck.checkArgs('fF', 'Globalization.getFirstDayOfWeek', arguments);
   exec(successCB, failureCB, "Globalization", "getFirstDayOfWeek", []);
},


/**
* Returns a number formatted as a string according to the client's user preferences.
* It returns the formatted number string to the successCB callback with a properties object as a
* parameter. If there is an error formatting the number, then the errorCB callback is invoked.
*
* The defaults are: type="decimal"
*
* @param {Number} number
* @param {Function} successCB
* @param {Function} errorCB
* @param {Object} options {optional}
*           type {String}: 'decimal', "percent", or 'currency'
*
* @return Object.value {String}: The formatted number string.
*
* @error GlobalizationError.FORMATTING_ERROR
*
* Example
*    globalization.numberToString(3.25,
*               function (number) {alert('number:' + number.value + '\n');},
*               function () {},
*               {type:'decimal'});
*/
numberToString:function(number, successCB, failureCB, options) {
   argscheck.checkArgs('nfFO', 'Globalization.numberToString', arguments);
   exec(successCB, failureCB, "Globalization", "numberToString", [{"number":
number, "options": options}]);
},
/**

* Parses a number formatted as a string according to the client's user preferences
and

* returns the corresponding number. It returns the number to the successCB
callback with a

* properties object as a parameter. If there is an error parsing the number string,
then

* the errorCB callback is invoked.

*

* The defaults are: type="decimal"

*

* @param {String} numberString

* @param {Function} successCB

* @param {Function} errorCB

* @param {Object} options {optional}

*           type {String}: 'decimal', "percent", or 'currency'

*

* @return Object.value {Number}: The parsed number.

*
```

```
* @error GlobalizationError.PARSING_ERROR
*
* Example
*    globalization.stringToNumber('1234.56',
*                function (number) {alert('Number:' + number.value + '\n');},
*                function () { alert('Error parsing number');});
*/
stringToNumber:function(numberString, successCB, failureCB, options) {
   argscheck.checkArgs('sfFO', 'Globalization.stringToNumber', arguments);
   exec(successCB, failureCB, "Globalization", "stringToNumber",
[{"numberString": numberString, "options": options}]);
},

/**
* Returns a pattern string for formatting and parsing numbers according to the
client's user
* preferences. It returns the pattern to the successCB callback with a properties
object as a
* parameter. If there is an error obtaining the pattern, then the errorCB callback is
invoked.
*
```

```
* The defaults are: type="decimal"

*

* @param {Function} successCB

* @param {Function} errorCB

* @param {Object} options {optional}

*         type {String}: 'decimal', "percent", or 'currency'

*

* @return      Object.pattern {String}: The number pattern for formatting and
parsing numbers.

*                               The patterns follow Unicode Technical Standard #35.

*                               http://unicode.org/reports/tr35/tr35-4.html

*            Object.symbol {String}: The symbol to be used when formatting and
parsing

*                               e.g., percent or currency symbol.

*           Object.fraction {Number}: The number of fractional digits to use when
parsing and

*                               formatting numbers.

*         Object.rounding {Number}: The rounding increment to use when parsing
and formatting.

*           Object.positive {String}: The symbol to use for positive numbers when
parsing and formatting.
```

```
*           Object.negative: {String}: The symbol to use for negative numbers when
parsing and formatting.

*             Object.decimal: {String}: The decimal symbol to use for parsing and
formatting.

*            Object.grouping: {String}: The grouping symbol to use for parsing and
formatting.

*

* @error GlobalizationError.PATTERN_ERROR

*

* Example

*    globalization.getNumberPattern(

*              function (pattern) {alert('Pattern:' + pattern.pattern + '\n');},

*              function () {});

*/

getNumberPattern:function(successCB, failureCB, options) {

   argscheck.checkArgs('fFO', 'Globalization.getNumberPattern', arguments);

   exec(successCB, failureCB, "Globalization", "getNumberPattern", [{"options":
options}]);

},


/**
```

```
* Returns a pattern string for formatting and parsing currency values according to the client's

* user preferences and ISO 4217 currency code. It returns the pattern to the successCB callback with a

* properties object as a parameter. If there is an error obtaining the pattern, then the errorCB

* callback is invoked.

*

* @param {String} currencyCode

* @param {Function} successCB

* @param {Function} errorCB

*

* @return      Object.pattern {String}: The currency pattern for formatting and parsing currency values.

*                          The patterns follow Unicode Technical Standard #35

*                          http://unicode.org/reports/tr35/tr35-4.html

*         Object.code {String}: The ISO 4217 currency code for the pattern.

*           Object.fraction {Number}: The number of fractional digits to use when parsing and

*                          formatting currency.
*          Object.rounding {Number}: The rounding increment to use when parsing and formatting.

*             Object.decimal: {String}: The decimal symbol to use for parsing and formatting.

*            Object.grouping: {String}: The grouping symbol to use for parsing and formatting.

*

* @error GlobalizationError.FORMATTING_ERROR

*

* Example

*    globalization.getCurrencyPattern('EUR',

*              function (currency) {alert('Pattern:' + currency.pattern + '\n');}

*              function () {});

*/

getCurrencyPattern:function(currencyCode, successCB, failureCB) {

   argscheck.checkArgs('sfF', 'Globalization.getCurrencyPattern', arguments);

   exec(successCB, failureCB, "Globalization", "getCurrencyPattern", [{"currencyCode": currencyCode}]);

}


};
```

```
module.exports = globalization;

});

// file: lib/common/plugin/globalization/symbols.js
define("cordova/plugin/globalization/symbols", function(require, exports,
module) {


var modulemapper = require('cordova/modulemapper');

modulemapper.clobbers('cordova/plugin/globalization', 'navigator.globalization');
modulemapper.clobbers('cordova/plugin/GlobalizationError',
'GlobalizationError');

});

// file: lib/android/plugin/inappbrowser/symbols.js
```

```
define("cordova/plugin/inappbrowser/symbols", function(require, exports, module) {


var modulemapper = require('cordova/modulemapper');

modulemapper.clobbers('cordova/plugin/InAppBrowser', 'open');

});

// file: lib/common/plugin/logger.js
define("cordova/plugin/logger", function(require, exports, module) {

//------------------------------------------------------------------------------
// The logger module exports the following properties/functions:
//
// LOG                          - constant for the level LOG
// ERROR                        - constant for the level ERROR
// WARN                         - constant for the level WARN
// INFO                         - constant for the level INFO
// DEBUG                      - constant for the level DEBUG
// logLevel()                 - returns current log level
// logLevel(value)            - sets and returns a new log level
// useConsole()               - returns whether logger is using console
// useConsole(value)          - sets and returns whether logger is using console
// log(message,...)           - logs a message at level LOG
// error(message,...)         - logs a message at level ERROR
// warn(message,...)          - logs a message at level WARN
// info(message,...)          - logs a message at level INFO
// debug(message,...)         - logs a message at level DEBUG
// logLevel(level,message,...)  - logs a message specified level
//
//------------------------------------------------------------------------------

var logger = exports;

var exec    = require('cordova/exec');
var utils   = require('cordova/utils');
var UseConsole   = true;
var UseLogger    = true;
var Queued       = [];
var DeviceReady  = false;
var CurrentLevel;

var originalConsole = console;

/**
 * Logging levels
 */

var Levels = [
    "LOG",
    "ERROR",
    "WARN",
    "INFO",
    "DEBUG"
];
```

```
/*
 * add the logging levels to the logger object and
 * to a separate levelsMap object for testing
 */

var LevelsMap = {};
for (var i=0; i<Levels.length; i++) {
   var level = Levels[i];
   LevelsMap[level] = i;
   logger[level]    = level;
}

CurrentLevel = LevelsMap.WARN;

/**
 * Getter/Setter for the logging level
 *
 * Returns the current logging level.
 *
 * When a value is passed, sets the logging level to that value.
 * The values should be one of the following constants:
 *    logger.LOG
 *    logger.ERROR
 *    logger.WARN
 *    logger.INFO
 *    logger.DEBUG
 *
 * The value used determines which messages get printed.  The logging
 * values above are in order, and only messages logged at the logging
 * level or above will actually be displayed to the user.  E.g., the
 * default level is WARN, so only messages logged with LOG, ERROR, or
 * WARN will be displayed; INFO and DEBUG messages will be ignored.
 */
logger.level = function (value) {
    if (arguments.length) {
        if (LevelsMap[value] === null) {
            throw new Error("invalid logging level: " + value);
        }
        CurrentLevel = LevelsMap[value];
    }

    return Levels[CurrentLevel];
};

/**
 * Getter/Setter for the useConsole functionality
 *
 * When useConsole is true, the logger will log via the
 * browser 'console' object.
 */
logger.useConsole = function (value) {
    if (arguments.length) UseConsole = !!value;

    if (UseConsole) {
        if (typeof console == "undefined") {
            throw new Error("global console object is not defined");
        }

        if (typeof console.log != "function") {
            throw new Error("global console object does not have a log function");
        }

        if (typeof console.useLogger == "function") {
            if (console.useLogger()) {
                throw new Error("console and logger are too intertwingly");
            }
        }
    }

    return UseConsole;
};

/**
 * Getter/Setter for the useLogger functionality
 *
 * When useLogger is true, the logger will log via the
 * native Logger plugin.
 */
logger.useLogger = function (value) {
    // Enforce boolean
    if (arguments.length) UseLogger = !!value;
    return UseLogger;
};

/**
 * Logs a message at the LOG level.
 *
 * Parameters passed after message are used applied to
 * the message with utils.format()
 */
logger.log   = function(message) { logWithArgs("LOG",   arguments); };

/**
 * Logs a message at the ERROR level.
 *
 * Parameters passed after message are used applied to
 * the message with utils.format()
 */
logger.error = function(message) { logWithArgs("ERROR", arguments); };

/**
 * Logs a message at the WARN level.
 *
 * Parameters passed after message are used applied to
 * the message with utils.format()
 */
logger.warn  = function(message) { logWithArgs("WARN",  arguments); };

/**
 * Logs a message at the INFO level.
 *
 * Parameters passed after message are used applied to
 * the message with utils.format()
 */
logger.info  = function(message) { logWithArgs("INFO",  arguments); };

/**
 * Logs a message at the DEBUG level.
 *
 * Parameters passed after message are used applied to
 * the message with utils.format()
 */
logger.debug = function(message) { logWithArgs("DEBUG", arguments); };

// log at the specified level with args
function logWithArgs(level, args) {
    args = [level].concat([].slice.call(args));
    logger.logLevel.apply(logger, args);
}

/**
 * Logs a message at the specified level.
 *
 * Parameters passed after message are used applied to
 * the message with utils.format()
 */
logger.logLevel = function(level /* , ... */) {
    // format the message with the parameters
    var formatArgs = [].slice.call(arguments, 1);
    var message    = logger.format.apply(logger.format, formatArgs);

    if (LevelsMap[level] === null) {
        throw new Error("invalid logging level: " + level);
    }

    if (LevelsMap[level] > CurrentLevel) return;

    // queue the message if not yet at deviceready
    if (!DeviceReady && !UseConsole) {
        Queued.push([level, message]);
        return;
```

```
}

// Log using the native logger if that is enabled
if (UseLogger) {
    exec(null, null, "Logger", "logLevel", [level, message]);
}

// Log using the console if that is enabled
if (UseConsole) {
    // make sure console is not using logger
    if (console.__usingCordovaLogger) {
        throw new Error("console and logger are too intertwingly");
    }

    // log to the console
    switch (level) {
        case logger.LOG:   originalConsole.log(message); break;
        case logger.ERROR: originalConsole.log("ERROR: " + message); break;
        case logger.WARN:  originalConsole.log("WARN: "  + message); break;
```

```
            case logger.INFO:  originalConsole.log("INFO: "  + message); break;
            case logger.DEBUG: originalConsole.log("DEBUG: " + message); break;
        }
    }
};


/**
 * Formats a string and arguments following it ala console.log()
 *
 * Any remaining arguments will be appended to the formatted string.
 *
 * for rationale, see FireBug's Console API:
 *    http://getfirebug.com/wiki/index.php/Console_API
 */
logger.format = function(formatString, args) {
    return __format(arguments[0], [].slice.call(arguments,1)).join(' ');
};
```

```
//-------------------------------------------------------------------------

/**

 * Formats a string and arguments following it ala vsprintf()

 *

 * format chars:

 *   %j - format arg as JSON

 *   %o - format arg as JSON

 *   %c - format arg as ''

 *   %% - replace with '%'

 * any other char following % will format it's

 * arg via toString().

 *

 * Returns an array containing the formatted string and any remaining

 * arguments.

 */

function __format(formatString, args) {

   if (formatString === null || formatString === undefined) return [""];

   if (arguments.length == 1) return [formatString.toString()];
   if (typeof formatString != "string")
       formatString = formatString.toString();

   var pattern = /(.*?)%(.)(.*)/;
   var rest    = formatString;
   var result  = [];

   while (args.length) {
       var match = pattern.exec(rest);
       if (!match) break;

       var arg   = args.shift();
       rest = match[3];
       result.push(match[1]);

       if (match[2] == '%') {
           result.push('%');
           args.unshift(arg);
```

```
            continue;
        }

        result.push(__formatted(arg, match[2]));
    }

    result.push(rest);

    var remainingArgs = [].slice.call(args);
    remainingArgs.unshift(result.join(''));
    return remainingArgs;
}

function __formatted(object, formatChar) {

    try {
        switch(formatChar) {
            case 'j':
            case 'o': return JSON.stringify(object);
            case 'c': return '';
        }
    }
    catch (e) {
        return "error JSON.stringify()ing argument: " + e;
    }

    if ((object === null) || (object === undefined)) {
        return Object.prototype.toString.call(object);
    }

    return object.toString();
}


//------------------------------------------------------------------------------
// when deviceready fires, log queued messages
logger.__onDeviceReady = function() {
    if (DeviceReady) return;
    DeviceReady = true;

    for (var i=0; i<Queued.length; i++) {
        var messageArgs = Queued[i];
        logger.logLevel(messageArgs[0], messageArgs[1]);
    }

    Queued = null;
};

// add a deviceready event to log queued messages
document.addEventListener("deviceready", logger.__onDeviceReady, false);

});

// file: lib/common/plugin/logger/symbols.js
define("cordova/plugin/logger/symbols", function(require, exports, module) {
```

```
var modulemapper = require('cordova/modulemapper');

modulemapper.clobbers('cordova/plugin/logger', 'cordova.logger');

});

// file: lib/android/plugin/media/symbols.js
define("cordova/plugin/media/symbols", function(require, exports, module) {


var modulemapper = require('cordova/modulemapper');

modulemapper.defaults('cordova/plugin/Media', 'Media');
modulemapper.clobbers('cordova/plugin/MediaError', 'MediaError');

});

// file: lib/common/plugin/network.js
```

```
define("cordova/plugin/network", function(require, exports, module) {


var exec = require('cordova/exec'),

    cordova = require('cordova'),

    channel = require('cordova/channel'),

    utils = require('cordova/utils');


// Link the onLine property with the Cordova-supplied network info.

// This works because we clobber the naviagtor object with our own

// object in bootstrap.js.

if (typeof navigator != 'undefined') {

    utils.defineGetter(navigator, 'onLine', function() {

        return this.connection.type != 'none';

    });

}


function NetworkConnection() {

    this.type = 'unknown';

}
```

```
/**
 * Get connection info
 *
 * @param {Function} successCallback The function to call when the Connection data is available
 * @param {Function} errorCallback The function to call when there is an error getting the Connection data. (OPTIONAL)
 */
NetworkConnection.prototype.getInfo = function(successCallback, errorCallback) {
   exec(successCallback, errorCallback, "NetworkStatus", "getConnectionInfo", []);
};

var me = new NetworkConnection();
var timerId = null;
var timeout = 500;

channel.onCordovaReady.subscribe(function() {
   me.getInfo(function(info) {
```

```
me.type = info;
if (info === "none") {
   // set a timer if still offline at the end of timer send the offline event
   timerId = setTimeout(function(){
      cordova.fireDocumentEvent("offline");
      timerId = null;
   }, timeout);
} else {
   // If there is a current offline event pending clear it
   if (timerId !== null) {
      clearTimeout(timerId);
      timerId = null;
   }
   cordova.fireDocumentEvent("online");
}

// should only fire this once
if (channel.onCordovaConnectionReady.state !== 2) {
   channel.onCordovaConnectionReady.fire();
```

```
        }
    },
    function (e) {
        // If we can't get the network info we should still tell Cordova
        // to fire the deviceready event.
        if (channel.onCordovaConnectionReady.state !== 2) {
            channel.onCordovaConnectionReady.fire();
        }
        console.log("Error initializing Network Connection: " + e);
    });
});

module.exports = me;

});

// file: lib/common/plugin/networkstatus/symbols.js
define("cordova/plugin/networkstatus/symbols", function(require, exports,
module) {
```

```
var modulemapper = require('cordova/modulemapper');

modulemapper.clobbers('cordova/plugin/network',
'navigator.network.connection', 'navigator.network.connection is deprecated. Use
navigator.connection instead.');
modulemapper.clobbers('cordova/plugin/network', 'navigator.connection');
modulemapper.defaults('cordova/plugin/Connection', 'Connection');

});

// file: lib/common/plugin/notification.js
define("cordova/plugin/notification", function(require, exports, module) {

var exec = require('cordova/exec');
var platform = require('cordova/platform');

/**
 * Provides access to notifications on the device.
 */
```

```
module.exports = {

    /**
     * Open a native alert dialog, with a customizable title and button text.
     *
     * @param {String} message              Message to print in the body of the alert
     * @param {Function} completeCallback    The callback that is called when user clicks on a button.
     * @param {String} title                Title of the alert dialog (default: Alert)
     * @param {String} buttonLabel          Label of the close button (default: OK)
     */
    alert: function(message, completeCallback, title, buttonLabel) {
        var _title = (title || "Alert");
        var _buttonLabel = (buttonLabel || "OK");
        exec(completeCallback, null, "Notification", "alert", [message, _title, _buttonLabel]);
    },

    /**
     * Open a native confirm dialog, with a customizable title and button text.

     * The result that the user selects is returned to the result callback.

     *

     * @param {String} message            Message to print in the body of the alert

     * @param {Function} resultCallback     The callback that is called when user
clicks on a button.

     * @param {String} title              Title of the alert dialog (default: Confirm)

     * @param {Array} buttonLabels             Array of the labels of the buttons
(default: ['OK', 'Cancel'])

     */

    confirm: function(message, resultCallback, title, buttonLabels) {

        var _title = (title || "Confirm");

        var _buttonLabels = (buttonLabels || ["OK", "Cancel"]);



        // Strings are deprecated!

        if (typeof _buttonLabels === 'string') {

           console.log("Notification.confirm(string, function, string, string) is
deprecated.  Use Notification.confirm(string, function, string, array).");

        }
     // Some platforms take an array of button label names.

     // Other platforms take a comma separated list.

     // For compatibility, we convert to the desired type based on the platform.

     if (platform.id == "android" || platform.id == "ios" || platform.id ==
"windowsphone" || platform.id == "blackberry10") {

        if (typeof _buttonLabels === 'string') {

          var buttonLabelString = _buttonLabels;

          _buttonLabels = _buttonLabels.split(","); // not crazy about changing
the var type here

        }

     } else {

        if (Array.isArray(_buttonLabels)) {

          var buttonLabelArray = _buttonLabels;

          _buttonLabels = buttonLabelArray.toString();

        }

     }

     exec(resultCallback, null, "Notification", "confirm", [message, _title,
_buttonLabels]);

  },
    /**
     * Open a native prompt dialog, with a customizable title and button text.
     * The following results are returned to the result callback:
     *  buttonIndex     Index number of the button selected.
     *  input1          The text entered in the prompt dialog box.
     *
     * @param {String} message                 Dialog message to display (default: "Prompt message")
     * @param {Function} resultCallback     The callback that is called when user clicks on a button.
     * @param {String} title             Title of the dialog (default: "Prompt")
     * @param {Array} buttonLabels          Array of strings for the button labels (default: ["OK","Cancel"])
     * @param {String} defaultText          Textbox input value (default: "Default text")
     */
    prompt: function(message, resultCallback, title, buttonLabels, defaultText) {
        var _message = (message || "Prompt message");
        var _title = (title || "Prompt");
        var _buttonLabels = (buttonLabels || ["OK","Cancel"]);
        var _defaultText = (defaultText || "Default text");
        exec(resultCallback, null, "Notification", "prompt", [_message, _title, _buttonLabels, _defaultText]);
    },

    /**
     * Causes the device to vibrate.
     *
     * @param {Integer} mills       The number of milliseconds to vibrate for.
     */
    vibrate: function(mills) {
        exec(null, null, "Notification", "vibrate", [mills]);
    },

    /**
     * Causes the device to beep.
     * On Android, the default notification ringtone is played "count" times.
     *
     * @param {Integer} count       The number of beeps.
     */
    beep: function(count) {
        exec(null, null, "Notification", "beep", [count]);
    }
};


});


// file: lib/android/plugin/notification/symbols.js
define("cordova/plugin/notification/symbols", function(require, exports, module)
{



var modulemapper = require('cordova/modulemapper');


modulemapper.clobbers('cordova/plugin/notification', 'navigator.notification');
modulemapper.merges('cordova/plugin/android/notification',
'navigator.notification');


});


// file: lib/common/plugin/requestFileSystem.js
```

```
define("cordova/plugin/requestFileSystem", function(require, exports, module) {

var argscheck = require('cordova/argscheck'),
    FileError = require('cordova/plugin/FileError'),
    FileSystem = require('cordova/plugin/FileSystem'),
    exec = require('cordova/exec');

/**
 * Request a file system in which to store application data.
 * @param type  local file system type
 * @param size  indicates how much storage space, in bytes, the application expects to need
 * @param successCallback  invoked with a FileSystem object
 * @param errorCallback  invoked if error occurs retrieving file system
 */
var requestFileSystem = function(type, size, successCallback, errorCallback) {
    argscheck.checkArgs('nnFF', 'requestFileSystem', arguments);
    var fail = function(code) {
        errorCallback && errorCallback(new FileError(code));
    };
```

```
if (type < 0 || type > 3) {
    fail(FileError.SYNTAX_ERR);
} else {
    // if successful, return a FileSystem object
    var success = function(file_system) {
        if (file_system) {
            if (successCallback) {
                // grab the name and root from the file system object
                var result = new FileSystem(file_system.name, file_system.root);
                successCallback(result);
            }
        }
        else {
            // no FileSystem object returned
            fail(FileError.NOT_FOUND_ERR);
        }
    };
    exec(success, fail, "File", "requestFileSystem", [type, size]);
    }
};

module.exports = requestFileSystem;

});

// file: lib/common/plugin/resolveLocalFileSystemURI.js
define("cordova/plugin/resolveLocalFileSystemURI", function(require, exports, module) {

var argscheck = require('cordova/argscheck'),
    DirectoryEntry = require('cordova/plugin/DirectoryEntry'),
    FileEntry = require('cordova/plugin/FileEntry'),
    FileError = require('cordova/plugin/FileError'),
    exec = require('cordova/exec');

/**
 * Look up file system Entry referred to by local URI.
 * @param {DOMString} uri  URI referring to a local file or directory
 * @param successCallback  invoked with Entry object corresponding to URI
 * @param errorCallback    invoked if error occurs retrieving file system entry
 */
module.exports = function(uri, successCallback, errorCallback) {
    argscheck.checkArgs('sFF', 'resolveLocalFileSystemURI', arguments);
    // error callback
    var fail = function(error) {
       errorCallback && errorCallback(new FileError(error));
    };
    // sanity check for 'not:valid:filename'
    if(!uri || uri.split(":").length > 2) {
       setTimeout( function() {
          fail(FileError.ENCODING_ERR);
       },0);
       return;
    }
    // if successful, return either a file or directory entry
    var success = function(entry) {
       var result;
```

```
        if (entry) {
            if (successCallback) {
                // create appropriate Entry object
                result = (entry.isDirectory) ? new DirectoryEntry(entry.name, entry.fullPath) : new FileEntry(entry.name, entry.fullPath);
                successCallback(result);
            }
        }
        else {
            // no Entry object returned
            fail(FileError.NOT_FOUND_ERR);
        }
    };

    exec(success, fail, "File", "resolveLocalFileSystemURI", [uri]);
};


});


// file: lib/common/plugin/splashscreen.js
```

```
define("cordova/plugin/splashscreen", function(require, exports, module) {

var exec = require('cordova/exec');

var splashscreen = {
    show:function() {
        exec(null, null, "SplashScreen", "show", []);
    },
    hide:function() {
        exec(null, null, "SplashScreen", "hide", []);
    }
};

module.exports = splashscreen;

});

// file: lib/common/plugin/splashscreen/symbols.js
define("cordova/plugin/splashscreen/symbols", function(require, exports, module) {
```

```
var modulemapper = require('cordova/modulemapper');


modulemapper.clobbers('cordova/plugin/splashscreen', 'navigator.splashscreen');


});


// file: lib/common/symbols.js
define("cordova/symbols", function(require, exports, module) {


var modulemapper = require('cordova/modulemapper');


// Use merges here in case others symbols files depend on this running first,
// but fail to declare the dependency with a require().
modulemapper.merges('cordova', 'cordova');
modulemapper.clobbers('cordova/exec', 'cordova.exec');
modulemapper.clobbers('cordova/exec', 'Cordova.exec');
```

```
});

// file: lib/common/utils.js
define("cordova/utils", function(require, exports, module) {

var utils = exports;

/**
 * Defines a property getter / setter for obj[key].
 */
utils.defineGetterSetter = function(obj, key, getFunc, opt_setFunc) {
    if (Object.defineProperty) {
        var desc = {
            get: getFunc,
            configurable: true
        };
        if (opt_setFunc) {
            desc.set = opt_setFunc;
        }
        Object.defineProperty(obj, key, desc);
    } else {
        obj.__defineGetter__(key, getFunc);
        if (opt_setFunc) {
            obj.__defineSetter__(key, opt_setFunc);
        }
    }
};

/**
 * Defines a property getter for obj[key].
 */
utils.defineGetter = utils.defineGetterSetter;

utils.arrayIndexOf = function(a, item) {
    if (a.indexOf) {
        return a.indexOf(item);
    }
    var len = a.length;
    for (var i = 0; i < len; ++i) {
        if (a[i] == item) {
            return i;
        }
    }
    return -1;
};


/**
 * Returns whether the item was found in the array.
 */
utils.arrayRemove = function(a, item) {
    var index = utils.arrayIndexOf(a, item);
    if (index != -1) {
        a.splice(index, 1);
    }
    return index != -1;
};
```

```
utils.typeName = function(val) {

   return Object.prototype.toString.call(val).slice(8, -1);

};


/**

 * Returns an indication of whether the argument is an array or not

 */

utils.isArray = function(a) {

   return utils.typeName(a) == 'Array';

};


/**

 * Returns an indication of whether the argument is a Date or not

 */

utils.isDate = function(d) {

   return utils.typeName(d) == 'Date';

};


/**
```

```
 * Does a deep clone of the object.
 */
utils.clone = function(obj) {
    if(!obj || typeof obj == 'function' || utils.isDate(obj) || typeof obj != 'object') {
        return obj;
    }

    var retVal, i;

    if(utils.isArray(obj)){
        retVal = [];
        for(i = 0; i < obj.length; ++i){
            retVal.push(utils.clone(obj[i]));
        }
        return retVal;
    }

    retVal = {};
    for(i in obj){
        if(!(i in retVal) || retVal[i] != obj[i]) {
            retVal[i] = utils.clone(obj[i]);
        }
    }
    return retVal;
};

/**
 * Returns a wrapped version of the function
 */
utils.close = function(context, func, params) {
    if (typeof params == 'undefined') {
        return function() {
            return func.apply(context, arguments);
        };
    } else {
        return function() {
            return func.apply(context, params);
        };
    }
};

/**
 * Create a UUID
 */
utils.createUUID = function() {
    return UUIDcreatePart(4) + '-' +
        UUIDcreatePart(2) + '-' +
        UUIDcreatePart(2) + '-' +
        UUIDcreatePart(2) + '-' +
        UUIDcreatePart(6);
};

/**
 * Extends a child object from a parent object using classical inheritance
 * pattern.
 */
utils.extend = (function() {
    // proxy used to establish prototype chain
    var F = function() {};
    // extend Child from Parent
    return function(Child, Parent) {
        F.prototype = Parent.prototype;
        Child.prototype = new F();
        Child.__super__ = Parent.prototype;
        Child.prototype.constructor = Child;
    };
}());

/**
 * Alerts a message in any available way: alert or console.log.
 */
utils.alert = function(msg) {
    if (window.alert) {
        window.alert(msg);
    } else if (console && console.log) {
        console.log(msg);
    }

};



//---------------------------------------------------------------------------

function UUIDcreatePart(length) {

    var uuidpart = "";

    for (var i=0; i<length; i++) {

        var uuidchar = parseInt((Math.random() * 256), 10).toString(16);

        if (uuidchar.length == 1) {

            uuidchar = "0" + uuidchar;

        }

        uuidpart += uuidchar;

    }

    return uuidpart;

}



});
```

```
window.cordova = require('cordova');
// file: lib/scripts/bootstrap.js

(function (context) {
    if (context._cordovaJsLoaded) {
        throw new Error('cordova.js included multiple times.');
    }
    context._cordovaJsLoaded = true;

    var channel = require('cordova/channel');
    var platformInitChannelsArray = [channel.onNativeReady, channel.onPluginsReady];

    function logUnfiredChannels(arr) {
        for (var i = 0; i < arr.length; ++i) {
            if (arr[i].state != 2) {
                console.log('Channel not fired: ' + arr[i].type);
            }
        }
    }

    window.setTimeout(function() {
        if (channel.onDeviceReady.state != 2) {
            console.log('deviceready has not fired after 5 seconds.');
            logUnfiredChannels(platformInitChannelsArray);
            logUnfiredChannels(channel.deviceReadyChannelsArray);
        }
    }, 5000);

    // Replace navigator before any modules are required(), to ensure it happens as soon as possible.
    // We replace it so that properties that can't be clobbered can instead be overridden.
    function replaceNavigator(origNavigator) {
        var CordovaNavigator = function() {};
        CordovaNavigator.prototype = origNavigator;
        var newNavigator = new CordovaNavigator();
        // This work-around really only applies to new APIs that are newer than Function.bind.
        // Without it, APIs such as getGamepads() break.
        if (CordovaNavigator.bind) {
            for (var key in origNavigator) {
                if (typeof origNavigator[key] == 'function') {
                    newNavigator[key] = origNavigator[key].bind(origNavigator);
                }
            }
        }
        return newNavigator;
    }
    if (context.navigator) {
        context.navigator = replaceNavigator(context.navigator);
    }

    // _nativeReady is global variable that the native side can set
    // to signify that the native code is ready. It is a global since
    // it may be called before any cordova JS is ready.
    if (window._nativeReady) {
        channel.onNativeReady.fire();
```

```
}

/**
 * Create all cordova objects once native side is ready.
 */
channel.join(function() {
   // Call the platform-specific initialization
   require('cordova/platform').initialize();

   // Fire event to notify that all objects are created
   channel.onCordovaReady.fire();

   // Fire onDeviceReady event once page has fully loaded, all
   // constructors have run and cordova info has been received from native
   // side.
   // This join call is deliberately made after platform.initialize() in
   // order that plugins may manipulate channel.deviceReadyChannelsArray
   // if necessary.
   channel.join(function() {
```

```
            require('cordova').fireDocumentEvent('deviceready');
        }, channel.deviceReadyChannelsArray);

    }, platformInitChannelsArray);

}(window));

// file: lib/scripts/bootstrap-android.js

require('cordova/channel').onNativeReady.fire();

// file: lib/scripts/plugin_loader.js

// Tries to load all plugins' js-modules.
// This is an async process, but onDeviceReady is blocked on onPluginsReady.
// onPluginsReady is fired when there are no plugins to load, or they are all done.
(function (context) {
    // To be populated with the handler by handlePluginsObject.
    var onScriptLoadingComplete;
    var scriptCounter = 0;
    function scriptLoadedCallback() {
        scriptCounter--;
        if (scriptCounter === 0) {
            onScriptLoadingComplete && onScriptLoadingComplete();
        }
    }

    function scriptErrorCallback(err) {
        // Open Question: If a script path specified in cordova_plugins.js does not exist, do we fail for all?
        // this is currently just continuing.
        scriptCounter--;
        if (scriptCounter === 0) {
            onScriptLoadingComplete && onScriptLoadingComplete();
        }
    }

    // Helper function to inject a <script> tag.
```

```
function injectScript(path) {

    scriptCounter++;

    var script = document.createElement("script");

    script.onload = scriptLoadedCallback;

    script.onerror = scriptErrorCallback;

    script.src = path;

    document.head.appendChild(script);

}


// Called when:

// * There are plugins defined and all plugins are finished loading.

// * There are no plugins to load.

function finishPluginLoading() {

    context.cordova.require('cordova/channel').onPluginsReady.fire();

}


// Handler for the cordova_plugins.js content.

// See plugman's plugin_loader.js for the details of this object.

// This function is only called if the really is a plugins array that isn't empty.
// Otherwise the onerror response handler will just call finishPluginLoading().
function handlePluginsObject(modules, path) {
    // First create the callback for when all plugins are loaded.
    var mapper = context.cordova.require('cordova/modulemapper');
    onScriptLoadingComplete = function() {
        // Loop through all the plugins and then through their clobbers and merges.
        for (var i = 0; i < modules.length; i++) {
            var module = modules[i];
            if (module) {
                try {
                    if (module.clobbers && module.clobbers.length) {
                        for (var j = 0; j < module.clobbers.length; j++) {
                            mapper.clobbers(module.id, module.clobbers[j]);
                        }
                    }

                    if (module.merges && module.merges.length) {
                        for (var k = 0; k < module.merges.length; k++) {
                            mapper.merges(module.id, module.merges[k]);
                            }
                        }

                        // Finally, if runs is truthy we want to simply require() the module.
                        // This can be skipped if it had any merges or clobbers, though,
                        // since the mapper will already have required the module.
                        if (module.runs && !(module.clobbers &&
module.clobbers.length) && !(module.merges && module.merges.length)) {
                            context.cordova.require(module.id);
                        }
                    }
                    catch(err) {
                        // error with module, most likely clobbers, should we continue?
                    }
                }
            }

            finishPluginLoading();
        };
```

```
    // Now inject the scripts.
    for (var i = 0; i < modules.length; i++) {
        injectScript(path + modules[i].file);
    }
}

// Find the root of the app
var path = '';
var scripts = document.getElementsByTagName('script');
var term = 'cordova.js';
for (var n = scripts.length-1; n>-1; n--) {
    var src = scripts[n].src;
    if (src.indexOf(term) == (src.length - term.length)) {
        path = src.substring(0, src.length - term.length);
        break;
    }
}

var plugins_json = path + 'cordova_plugins.json';
```

```
    var plugins_js = path + 'cordova_plugins.js';


    // One some phones (Windows) this xhr.open throws an Access Denied exception
    // So lets keep trying, but with a script tag injection technique instead of XHR
    var injectPluginScript = function injectPluginScript() {
        try {
            var script = document.createElement("script");
            script.onload = function(){
                var list = cordova.require("cordova/plugin_list");
                handlePluginsObject(list,path);
            };
            script.onerror = function() {
                // Error loading cordova_plugins.js, file not found or something
                // this is an acceptable error, pre-3.0.0, so we just move on.
                finishPluginLoading();
            };
            script.src = plugins_js;
            document.head.appendChild(script);
    } catch(err){
      finishPluginLoading();
    }
  }


  // Try to XHR the cordova_plugins.json file asynchronously.
  var xhr = new XMLHttpRequest();
  xhr.onload = function() {
    // If the response is a JSON string which composes an array, call
handlePluginsObject.
    // If the request fails, or the response is not a JSON array, just call
finishPluginLoading.
    var obj;
    try {
      obj = (this.status == 0 || this.status == 200) && this.responseText &&
JSON.parse(this.responseText);
    } catch (err) {
      // obj will be undefined.
    }
```

```
        if (Array.isArray(obj) && obj.length > 0) {
            handlePluginsObject(obj, path);
        } else {
            finishPluginLoading();
        }
    };
    xhr.onerror = function() {
        // In this case, the json file was not present, but XHR was allowed,
        // so we should still try the script injection technique with the js file
        // in case that is there.
        injectPluginScript();
    };
    try { // we commented we were going to try, so let us actually try and catch
        xhr.open('GET', plugins_json, true); // Async
        xhr.send();
    } catch(err){
        injectPluginScript();
    }
}(window));
```

```
})();
```

10. assets/www/cordova-plugins.js

```
cordova.define('cordova/plugin_list', function(require, exports, module) {
module.exports = []
});
```

11. assets/www/index.html

```
<!DOCTYPE html>
<!--
   Licensed to the Apache Software Foundation (ASF) under one
   or more contributor license agreements.  See the NOTICE file
   distributed with this work for additional information
   regarding copyright ownership.  The ASF licenses this file
   to you under the Apache License, Version 2.0 (the
   "License"); you may not use this file except in compliance
   with the License.  You may obtain a copy of the License at
```

```
    http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing,
    software distributed under the License is distributed on an
    "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
     KIND, either express or implied.  See the License for the
    specific language governing permissions and limitations
    under the License.
-->
<html>
    <head>
        <meta charset="utf-8" />
        <meta name="format-detection" content="telephone=no" />
        <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1, minimum-scale=1, width=device-width, height=device-height, target-densitydpi=device-dpi" />
        <link rel="stylesheet" type="text/css" href="css/index.css" />
        <title>Hello World</title>
    </head>
    <body>
        <div class="app">
            <h1>PhoneGap</h1>
            <div id="deviceready" class="blink">
                <p class="event listening">Connecting to Device</p>
                <p class="event received">Device is Ready</p>
            </div>
        </div>
        <script type="text/javascript" src="phonegap.js"></script>
        <script type="text/javascript" src="js/calendar.js"></script>
        <script type="text/javascript" src="js/index.js"></script>
        <script type="text/javascript">
            app.initialize();
        </script>
    </body>
</html>
```

12. assets/www/spec.html

```
<!DOCTYPE html>
```

```
<html>
```

```
 <head>
   <title>Jasmine Spec Runner</title>

   <!-- jasmine source -->
   <link rel="shortcut icon" type="image/png" href="spec/lib/jasmine-1.2.0/jasmine_favicon.png">
   <link rel="stylesheet" type="text/css" href="spec/lib/jasmine-1.2.0/jasmine.css">
   <script type="text/javascript" src="spec/lib/jasmine-1.2.0/jasmine.js"></script>
   <script type="text/javascript" src="spec/lib/jasmine-1.2.0/jasmine-html.js"></script>

   <!-- include source files here... -->
   <script type="text/javascript" src="js/index.js"></script>

   <!-- include spec files here... -->
   <script type="text/javascript" src="spec/helper.js"></script>
   <script type="text/javascript" src="spec/index.js"></script>

   <script type="text/javascript">
```

```
(function() {
    var jasmineEnv = jasmine.getEnv();
    jasmineEnv.updateInterval = 1000;

    var htmlReporter = new jasmine.HtmlReporter();

    jasmineEnv.addReporter(htmlReporter);

    jasmineEnv.specFilter = function(spec) {
        return htmlReporter.specFilter(spec);
    };

    var currentWindowOnload = window.onload;

    window.onload = function() {
        if (currentWindowOnload) {
            currentWindowOnload();
        }
        execJasmine();
```

```
                };


                function execJasmine() {
                    jasmineEnv.execute();
                }
            })();
        </script>
    </head>
    <body>
        <div id="stage" style="display:none;"></div>
    </body>
</html>
```

13. assets/www/css/index.css

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */
* {
   -webkit-tap-highlight-color: rgba(0,0,0,0); /* make transparent link selection, adjust last value opacity 0 to 1.0 */
}

body {
   -webkit-touch-callout: none;                  /* prevent callout to copy image, etc
when tap to hold */
   -webkit-text-size-adjust: none;               /* prevent webkit from resizing text to
fit */
   -webkit-user-select: none;                    /* prevent copy paste, to allow, change
'none' to 'text' */
   background-color:#E4E4E4;
   background-image:linear-gradient(top, #A7A7A7 0%, #E4E4E4 51%);
   background-image:-webkit-linear-gradient(top, #A7A7A7 0%, #E4E4E4 51%);
   background-image:-ms-linear-gradient(top, #A7A7A7 0%, #E4E4E4 51%);
   background-image:-webkit-gradient(
      linear,
      left top,
      left bottom,
      color-stop(0, #A7A7A7),
      color-stop(0.51, #E4E4E4)
   );
   background-attachment:fixed;
   font-family:'HelveticaNeue-Light', 'HelveticaNeue', Helvetica, Arial, sans-serif;
   font-size:12px;
```

```
    height:100%;
    margin:0px;
    padding:0px;
    text-transform:uppercase;
    width:100%;
}

/* Portrait layout (default) */
.app {
    background:url(../img/logo.png) no-repeat center top; /* 170px x 200px */
    position:absolute;             /* position in the center of the screen */
    left:50%;
    top:50%;
    height:50px;                   /* text area height */
    width:225px;                   /* text area width */
    text-align:center;
    padding:180px 0px 0px 0px;     /* image height is 200px (bottom 20px are overlapped with text) */
    margin:-115px 0px 0px -112px;  /* offset vertical: half of image height and text area height */
```

```
                    /* offset horizontal: half of text area width */
}


/* Landscape layout (with min-width) */
@media screen and (min-aspect-ratio: 1/1) and (min-width:400px) {
    .app {
        background-position:left center;
        padding:75px 0px 75px 170px;  /* padding-top + padding-bottom + text area
= image height */
        margin:-90px 0px 0px -198px;  /* offset vertical: half of image height */
                    /* offset horizontal: half of image width and text area
width */
    }
}


h1 {
    font-size:24px;
    font-weight:normal;
    margin:0px;
    overflow:visible;
    padding:0px;
    text-align:center;
}

.event {
    border-radius:4px;
    -webkit-border-radius:4px;
    color:#FFFFFF;
    font-size:12px;
    margin:0px 30px;
    padding:2px 0px;
}

.event.listening {
    background-color:#333333;
    display:block;
}

.event.received {
    background-color:#4B946A;

    display:none;

}


@keyframes fade {

    from { opacity: 1.0; }

    50% { opacity: 0.4; }

    to { opacity: 1.0; }

}


@-webkit-keyframes fade {

    from { opacity: 1.0; }

    50% { opacity: 0.4; }

    to { opacity: 1.0; }

}


.blink {

    animation:fade 3000ms infinite;

    -webkit-animation:fade 3000ms infinite;
```

```
}
```

14. assets/www/js/calendar.js

```
var calendarPlugin = {
  createEvent: function(title, location, notes, startDate, endDate,
successCallback, errorCallback) {
      cordova.exec(
         successCallback, // success callback function
         errorCallback, // error callback function
         'CalendarPlugin', // mapped to our native Java class called
"CalendarPlugin"
         'addCalendarEntry', // with this action name
         [{              // and this array of custom arguments to create our entry
            "title": title,
            "description": notes,
            "eventLocation": location,
            "startTimeMillis": startDate.getTime(),
            "endTimeMillis": endDate.getTime()
         }]
      );
  }
}
```

15. assets/www/js/index.js

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */
var app = {
    // Application Constructor
    initialize: function() {
        this.bindEvents();
    },
    // Bind Event Listeners
    //
    // Bind any events that are required on startup. Common events are:
    // 'load', 'deviceready', 'offline', and 'online'.
    bindEvents: function() {
        document.addEventListener('deviceready', this.onDeviceReady, false);
    },
    // deviceready Event Handler
    //
    // The scope of 'this' is the event. In order to call the 'receivedEvent'
    // function, we must explicity call 'app.receivedEvent(...);'
```

```
onDeviceReady: function() {
    app.receivedEvent('deviceready');
},
// Update DOM on a Received Event
receivedEvent: function(id) {
    var parentElement = document.getElementById(id);
    var listeningElement = parentElement.querySelector('.listening');
    var receivedElement = parentElement.querySelector('.received');

    listeningElement.setAttribute('style', 'display:none;');
    receivedElement.setAttribute('style', 'display:block;');

    console.log('Received Event: ' + id);
    app.addToCal();
},
addToCal: function() {
        var startDate = new Date("July 19, 2013 8:00:00");
        var endDate = new Date("July 19, 2013 4:00:00");
        var title = "PhoneGap Day";
        var location = "Portland";

        var notes = "Arrive on time, don't want to miss out!";

        var success = function() { alert("Success"); };

        var error = function(message) { alert("Oopsie! " + message); };

        calendarPlugin.createEvent(title, location, notes, startDate, endDate, success, error);

    },

};
```

16. assets/www/spec/helper.js

```
/*

 * Licensed to the Apache Software Foundation (ASF) under one

 * or more contributor license agreements.  See the NOTICE file

 * distributed with this work for additional information

 * regarding copyright ownership.  The ASF licenses this file

 * to you under the Apache License, Version 2.0 (the

 * "License"); you may not use this file except in compliance

 * with the License.  You may obtain a copy of the License at

 *

 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */
afterEach(function() {
    document.getElementById('stage').innerHTML = '';
});


var helper = {
    trigger: function(obj, name) {
        var e = document.createEvent('Event');
        e.initEvent(name, true, true);
        obj.dispatchEvent(e);
    },
    getComputedStyle: function(querySelector, property) {
        var element = document.querySelector(querySelector);
        return window.getComputedStyle(element).getPropertyValue(property);
    }
};
```

17. assets/www/spec/index.js

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */
describe('app', function() {
    describe('initialize', function() {
        it('should bind deviceready', function() {
            runs(function() {
                spyOn(app, 'onDeviceReady');
                app.initialize();
                helper.trigger(window.document, 'deviceready');
            });

            waitsFor(function() {
                return (app.onDeviceReady.calls.length > 0);
            }, 'onDeviceReady should be called once', 500);

            runs(function() {
```

```
            expect(app.onDeviceReady).toHaveBeenCalled();
        });
    });
});

describe('onDeviceReady', function() {
    it('should report that it fired', function() {
        spyOn(app, 'receivedEvent');
        app.onDeviceReady();
        expect(app.receivedEvent).toHaveBeenCalledWith('deviceready');
    });
});

describe('receivedEvent', function() {
    beforeEach(function() {
        var el = document.getElementById('stage');
        el.innerHTML = ['<div id="deviceready">',
                        '    <p class="event listening">Listening</p>',
                        '    <p class="event received">Received</p>',
                '</div>'].join('\n');
        });

        it('should hide the listening element', function() {
            app.receivedEvent('deviceready');
            var displayStyle = helper.getComputedStyle('#deviceready .listening', 'display');
            expect(displayStyle).toEqual('none');
        });

        it('should show the received element', function() {
            app.receivedEvent('deviceready');
            var displayStyle = helper.getComputedStyle('#deviceready .received', 'display');
            expect(displayStyle).toEqual('block');
        });
    });
});
```

18. assets/www/spec/lib/jasmine-1.2.0/jasmine.css

```
body { background-color: #eeeeee; padding: 0; margin: 5px; overflow-y: scroll; }

#HTMLReporter { font-size: 11px; font-family: Monaco, "Lucida Console", monospace; line-height: 14px; color: #333333; }
#HTMLReporter a { text-decoration: none; }
#HTMLReporter a:hover { text-decoration: underline; }
#HTMLReporter p, #HTMLReporter h1, #HTMLReporter h2, #HTMLReporter h3, #HTMLReporter h4, #HTMLReporter h5, #HTMLReporter h6 { margin: 0; line-height: 14px; }
#HTMLReporter .banner, #HTMLReporter .symbolSummary, #HTMLReporter .summary, #HTMLReporter .resultMessage, #HTMLReporter .specDetail .description, #HTMLReporter .alert .bar, #HTMLReporter .stackTrace { padding-left: 9px; padding-right: 9px; }
#HTMLReporter #jasmine_content { position: fixed; right: 100%; }
#HTMLReporter .version { color: #aaaaaa; }
#HTMLReporter .banner { margin-top: 14px; }
#HTMLReporter .duration { color: #aaaaaa; float: right; }
#HTMLReporter .symbolSummary { overflow: hidden; *zoom: 1; margin: 14px 0; }
#HTMLReporter .symbolSummary li { display: block; float: left; height: 7px; width: 14px; margin-bottom: 7px; font-size: 16px; }
#HTMLReporter .symbolSummary li.passed { font-size: 14px; }
```

```
#HTMLReporter .symbolSummary li.passed:before { color: #5e7d00; content: "\02022"; }

#HTMLReporter .symbolSummary li.failed { line-height: 9px; }

#HTMLReporter .symbolSummary li.failed:before { color: #b03911; content: "x"; font-weight: bold; margin-left: -1px; }

#HTMLReporter .symbolSummary li.skipped { font-size: 14px; }

#HTMLReporter .symbolSummary li.skipped:before { color: #bababa; content: "\02022"; }

#HTMLReporter .symbolSummary li.pending { line-height: 11px; }

#HTMLReporter .symbolSummary li.pending:before { color: #aaaaaa; content: "-"; }

#HTMLReporter .bar { line-height: 28px; font-size: 14px; display: block; color: #eee; }

#HTMLReporter .runningAlert { background-color: #666666; }

#HTMLReporter .skippedAlert { background-color: #aaaaaa; }

#HTMLReporter .skippedAlert:first-child { background-color: #333333; }

#HTMLReporter .skippedAlert:hover { text-decoration: none; color: white; text-decoration: underline; }

#HTMLReporter .passingAlert { background-color: #a6b779; }

#HTMLReporter .passingAlert:first-child { background-color: #5e7d00; }

#HTMLReporter .failingAlert { background-color: #cf867e; }
```

```
#HTMLReporter .failingAlert:first-child { background-color: #b03911; }

#HTMLReporter .results { margin-top: 14px; }

#HTMLReporter #details { display: none; }

#HTMLReporter .resultsMenu, #HTMLReporter .resultsMenu a { background-color: #fff; color: #333333; }

#HTMLReporter.showDetails .summaryMenuItem { font-weight: normal; text-decoration: inherit; }

#HTMLReporter.showDetails .summaryMenuItem:hover { text-decoration: underline; }

#HTMLReporter.showDetails .detailsMenuItem { font-weight: bold; text-decoration: underline; }

#HTMLReporter.showDetails .summary { display: none; }

#HTMLReporter.showDetails #details { display: block; }

#HTMLReporter .summaryMenuItem { font-weight: bold; text-decoration: underline; }

#HTMLReporter .summary { margin-top: 14px; }

#HTMLReporter .summary .suite .suite, #HTMLReporter .summary .specSummary { margin-left: 14px; }

#HTMLReporter .summary .specSummary.passed a { color: #5e7d00; }

#HTMLReporter .summary .specSummary.failed a { color: #b03911; }

#HTMLReporter .description + .suite { margin-top: 0; }
```

```
#HTMLReporter .suite { margin-top: 14px; }

#HTMLReporter .suite a { color: #333333; }

#HTMLReporter #details .specDetail { margin-bottom: 28px; }

#HTMLReporter #details .specDetail .description { display: block; color: white; background-color: #b03911; }

#HTMLReporter .resultMessage { padding-top: 14px; color: #333333; }

#HTMLReporter .resultMessage span.result { display: block; }

#HTMLReporter .stackTrace { margin: 5px 0 0 0; max-height: 224px; overflow: auto; line-height: 18px; color: #666666; border: 1px solid #ddd; background: white; white-space: pre; }


#TrivialReporter { padding: 8px 13px; position: absolute; top: 0; bottom: 0; left: 0; right: 0; overflow-y: scroll; background-color: white; font-family: "Helvetica Neue Light", "Lucida Grande", "Calibri", "Arial", sans-serif; /*.resultMessage {*/ /*white-space: pre;*/ /*}*/ }

#TrivialReporter a:visited, #TrivialReporter a { color: #303; }

#TrivialReporter a:hover, #TrivialReporter a:active { color: blue; }

#TrivialReporter .run_spec { float: right; padding-right: 5px; font-size: .8em; text-decoration: none; }

#TrivialReporter .banner { color: #303; background-color: #fef; padding: 5px; }

#TrivialReporter .logo { float: left; font-size: 1.1em; padding-left: 5px; }

#TrivialReporter .logo .version { font-size: .6em; padding-left: 1em; }
```

```css
#TrivialReporter .runner.running { background-color: yellow; }

#TrivialReporter .options { text-align: right; font-size: .8em; }

#TrivialReporter .suite { border: 1px outset gray; margin: 5px 0; padding-left: 1em; }

#TrivialReporter .suite .suite { margin: 5px; }

#TrivialReporter .suite.passed { background-color: #dfd; }

#TrivialReporter .suite.failed { background-color: #fdd; }

#TrivialReporter .spec { margin: 5px; padding-left: 1em; clear: both; }

#TrivialReporter .spec.failed, #TrivialReporter .spec.passed, #TrivialReporter .spec.skipped { padding-bottom: 5px; border: 1px solid gray; }

#TrivialReporter .spec.failed { background-color: #fbb; border-color: red; }

#TrivialReporter .spec.passed { background-color: #bfb; border-color: green; }

#TrivialReporter .spec.skipped { background-color: #bbb; }

#TrivialReporter .messages { border-left: 1px dashed gray; padding-left: 1em; padding-right: 1em; }

#TrivialReporter .passed { background-color: #cfc; display: none; }

#TrivialReporter .failed { background-color: #fbb; }

#TrivialReporter .skipped { color: #777; background-color: #eee; display: none; }

#TrivialReporter .resultMessage span.result { display: block; line-height: 2em; color: black; }

#TrivialReporter .resultMessage .mismatch { color: black; }
```

```
#TrivialReporter .stackTrace { white-space: pre; font-size: .8em; margin-left: 10px; max-height: 5em; overflow: auto; border: 1px inset red; padding: 1em; background: #eef; }

#TrivialReporter .finished-at { padding-left: 1em; font-size: .6em; }

#TrivialReporter.show-passed .passed, #TrivialReporter.show-skipped .skipped { display: block; }

#TrivialReporter #jasmine_content { position: fixed; right: 100%; }

#TrivialReporter .runner { border: 1px solid gray; display: block; margin: 5px 0; padding: 2px 0 2px 10px; }
```

19. assets/www/spec/lib/jasmine-1.2.0/jasmine.js

```
var isCommonJS = typeof window == "undefined";


/**
 * Top level namespace for Jasmine, a lightweight JavaScript BDD/spec/testing framework.
 *
 * @namespace
 */
var jasmine = {};
if (isCommonJS) exports.jasmine = jasmine;
```

```
/**
 * @private
 */
jasmine.unimplementedMethod_ = function() {
  throw new Error("unimplemented method");
};

/**
 * Use <code>jasmine.undefined</code> instead of <code>undefined</code>, since <code>undefined</code> is just
 * a plain old variable and may be redefined by somebody else.
 *
 * @private
 */
jasmine.undefined = jasmine.___undefined___;

/**
 * Show diagnostic messages in the console if set to true
 *
 */
```

```
jasmine.VERBOSE = false;


/**
 * Default interval in milliseconds for event loop yields (e.g. to allow network activity or to refresh the screen with the HTML-based runner). Small values here may result in slow test running. Zero means no updates until all tests have completed.
 *
 */
jasmine.DEFAULT_UPDATE_INTERVAL = 250;


/**
 * Default timeout interval in milliseconds for waitsFor() blocks.
 */
jasmine.DEFAULT_TIMEOUT_INTERVAL = 5000;


jasmine.getGlobal = function() {
  function getGlobal() {
    return this;
  }
  return getGlobal();
};


/**
 * Allows for bound functions to be compared.  Internal use only.
 *
 * @ignore
 * @private
 * @param base {Object} bound 'this' for the function
 * @param name {Function} function to find
 */
jasmine.bindOriginal_ = function(base, name) {
  var original = base[name];
  if (original.apply) {
    return function() {
      return original.apply(base, arguments);
    };
  } else {
    // IE support
    return jasmine.getGlobal()[name];
  }
};

jasmine.setTimeout = jasmine.bindOriginal_(jasmine.getGlobal(), 'setTimeout');
jasmine.clearTimeout = jasmine.bindOriginal_(jasmine.getGlobal(),
'clearTimeout');
jasmine.setInterval = jasmine.bindOriginal_(jasmine.getGlobal(), 'setInterval');
jasmine.clearInterval = jasmine.bindOriginal_(jasmine.getGlobal(),
'clearInterval');

jasmine.MessageResult = function(values) {
  this.type = 'log';
  this.values = values;
  this.trace = new Error(); // todo: test better
};

jasmine.MessageResult.prototype.toString = function() {
  var text = "";
  for (var i = 0; i < this.values.length; i++) {
    if (i > 0) text += " ";
    if (jasmine.isString_(this.values[i])) {
      text += this.values[i];
    } else {
      text += jasmine.pp(this.values[i]);
    }
  }
  return text;
};

jasmine.ExpectationResult = function(params) {
  this.type = 'expect';
  this.matcherName = params.matcherName;
  this.passed_ = params.passed;
  this.expected = params.expected;
  this.actual = params.actual;
  this.message = this.passed_ ? 'Passed.' : params.message;
  var trace = (params.trace || new Error(this.message));

  this.trace = this.passed_ ? '' : trace;

};


jasmine.ExpectationResult.prototype.toString = function () {

  return this.message;

};


jasmine.ExpectationResult.prototype.passed = function () {

  return this.passed_;

};


/**

 * Getter for the Jasmine environment. Ensures one gets created

 */

jasmine.getEnv = function() {

  var env = jasmine.currentEnv_ = jasmine.currentEnv_ || new jasmine.Env();

  return env;

};
```

```
/**
 * @ignore
 * @private
 * @param value
 * @returns {Boolean}
 */
jasmine.isArray_ = function(value) {
  return jasmine.isA_("Array", value);
};

/**
 * @ignore
 * @private
 * @param value
 * @returns {Boolean}
 */
jasmine.isString_ = function(value) {
  return jasmine.isA_("String", value);
};

/**
 * @ignore
 * @private
 * @param value
 * @returns {Boolean}
 */
jasmine.isNumber_ = function(value) {
  return jasmine.isA_("Number", value);
};

/**
 * @ignore
 * @private
 * @param {String} typeName
 * @param value
 * @returns {Boolean}
 */
```

```
jasmine.isA_ = function(typeName, value) {
  return Object.prototype.toString.apply(value) === '[object ' + typeName + ']';
};

/**
 * Pretty printer for expecations.  Takes any object and turns it into a human-readable string.
 *
 * @param value {Object} an object to be outputted
 * @returns {String}
 */
jasmine.pp = function(value) {
  var stringPrettyPrinter = new jasmine.StringPrettyPrinter();
  stringPrettyPrinter.format(value);
  return stringPrettyPrinter.string;
};

/**
 * Returns true if the object is a DOM Node.
 *
 * @param {Object} obj object to check
 * @returns {Boolean}
 */
jasmine.isDomNode = function(obj) {
  return obj.nodeType > 0;
};

/**
 * Returns a matchable 'generic' object of the class type.  For use in expecations of type when values don't matter.
 *
 * @example
 * // don't care about which function is passed in, as long as it's a function
 * expect(mySpy).toHaveBeenCalledWith(jasmine.any(Function));
 *
 * @param {Class} clazz
 * @returns matchable object of the type clazz
 */
jasmine.any = function(clazz) {
  return new jasmine.Matchers.Any(clazz);
```

```
};

/**
 * Returns a matchable subset of a JSON object. For use in expectations when you don't care about all of the
 * attributes on the object.
 *
 * @example
 * // don't care about any other attributes than foo.
 *   expect(mySpy).toHaveBeenCalledWith(jasmine.objectContaining({foo: "bar"});
 *
 * @param sample {Object} sample
 * @returns matchable object for the sample
 */
jasmine.objectContaining = function (sample) {
   return new jasmine.Matchers.ObjectContaining(sample);
};

/**
```

```
 * Jasmine Spies are test doubles that can act as stubs, spies, fakes or when used in an expecation, mocks.
 *
 * Spies should be created in test setup, before expectations.  They can then be checked, using the standard Jasmine
 * expectation syntax. Spies can be checked if they were called or not and what the calling params were.
 *
 * A Spy has the following fields: wasCalled, callCount, mostRecentCall, and argsForCall (see docs).
 *
 * Spies are torn down at the end of every spec.
 *
 * Note: Do <b>not</b> call new jasmine.Spy() directly - a spy must be created using spyOn, jasmine.createSpy or jasmine.createSpyObj.
 *
 * @example
 * // a stub
 * var myStub = jasmine.createSpy('myStub');  // can be used anywhere
 *
 * // spy example
```

```
* var foo = {
*   not: function(bool) { return !bool; }
* }
*
* // actual foo.not will not be called, execution stops
* spyOn(foo, 'not');

// foo.not spied upon, execution will continue to implementation
* spyOn(foo, 'not').andCallThrough();
*
* // fake example
* var foo = {
*   not: function(bool) { return !bool; }
* }
*
* // foo.not(val) will return val
* spyOn(foo, 'not').andCallFake(function(value) {return value;});
*
* // mock example
```

```
 * foo.not(7 == 7);
 * expect(foo.not).toHaveBeenCalled();
 * expect(foo.not).toHaveBeenCalledWith(true);
 *
 * @constructor
 * @see spyOn, jasmine.createSpy, jasmine.createSpyObj
 * @param {String} name
 */
jasmine.Spy = function(name) {
  /**
   * The name of the spy, if provided.
   */
  this.identity = name || 'unknown';
  /**
   *  Is this Object a spy?
   */
  this.isSpy = true;
  /**
   * The actual function this spy stubs.
 */
this.plan = function() {
};
/**
 * Tracking of the most recent call to the spy.
 * @example
 * var mySpy = jasmine.createSpy('foo');
 * mySpy(1, 2);
 * mySpy.mostRecentCall.args = [1, 2];
 */
this.mostRecentCall = {};

/**
 * Holds arguments for each call to the spy, indexed by call count
 * @example
 * var mySpy = jasmine.createSpy('foo');
 * mySpy(1, 2);
 * mySpy(7, 8);
 * mySpy.mostRecentCall.args = [7, 8];
```

```
   * mySpy.argsForCall[0] = [1, 2];
   * mySpy.argsForCall[1] = [7, 8];
   */
  this.argsForCall = [];
  this.calls = [];
};

/**
 * Tells a spy to call through to the actual implemenatation.
 *
 * @example
 * var foo = {
 *   bar: function() { // do some stuff }
 * }
 *
 * // defining a spy on an existing property: foo.bar
 * spyOn(foo, 'bar').andCallThrough();
 */
jasmine.Spy.prototype.andCallThrough = function() {
  this.plan = this.originalValue;
  return this;
};

/**
 * For setting the return value of a spy.
 *
 * @example
 * // defining a spy from scratch: foo() returns 'baz'
 * var foo = jasmine.createSpy('spy on foo').andReturn('baz');
 *
 * // defining a spy on an existing property: foo.bar() returns 'baz'
 * spyOn(foo, 'bar').andReturn('baz');
 *
 * @param {Object} value
 */
jasmine.Spy.prototype.andReturn = function(value) {
  this.plan = function() {
    return value;
  };
  return this;
};

/**
 * For throwing an exception when a spy is called.
 *
 * @example
 * // defining a spy from scratch: foo() throws an exception w/ message 'ouch'
 * var foo = jasmine.createSpy('spy on foo').andThrow('baz');
 *
 * // defining a spy on an existing property: foo.bar() throws an exception w/ message 'ouch'
 * spyOn(foo, 'bar').andThrow('baz');
 *
 * @param {String} exceptionMsg
 */
jasmine.Spy.prototype.andThrow = function(exceptionMsg) {
  this.plan = function() {
    throw exceptionMsg;
  };
  return this;
};

/**
 * Calls an alternate implementation when a spy is called.
 *
 * @example
 * var baz = function() {
 *   // do some stuff, return something
 * }
 * // defining a spy from scratch: foo() calls the function baz
 * var foo = jasmine.createSpy('spy on foo').andCall(baz);
 *
 * // defining a spy on an existing property: foo.bar() calls an anonymnous function
 * spyOn(foo, 'bar').andCall(function() { return 'baz';} );
 *
 * @param {Function} fakeFunc
 */
```

```
jasmine.Spy.prototype.andCallFake = function(fakeFunc) {
  this.plan = fakeFunc;
  return this;
};

/**
 * Resets all of a spy's the tracking variables so that it can be used again.
 *
 * @example
 * spyOn(foo, 'bar');
 *
 * foo.bar();
 *
 * expect(foo.bar.callCount).toEqual(1);
 *
 * foo.bar.reset();
 *
 * expect(foo.bar.callCount).toEqual(0);
 */
```

```
jasmine.Spy.prototype.reset = function() {
  this.wasCalled = false;
  this.callCount = 0;
  this.argsForCall = [];
  this.calls = [];
  this.mostRecentCall = {};
};

jasmine.createSpy = function(name) {

  var spyObj = function() {
    spyObj.wasCalled = true;
    spyObj.callCount++;
    var args = jasmine.util.argsToArray(arguments);
    spyObj.mostRecentCall.object = this;
    spyObj.mostRecentCall.args = args;
    spyObj.argsForCall.push(args);
    spyObj.calls.push({object: this, args: args});
    return spyObj.plan.apply(this, arguments);
  };

  var spy = new jasmine.Spy(name);

  for (var prop in spy) {
    spyObj[prop] = spy[prop];
  }

  spyObj.reset();

  return spyObj;
};

/**
 * Determines whether an object is a spy.
 *
 * @param {jasmine.Spy|Object} putativeSpy
 * @returns {Boolean}
 */
```

```
jasmine.isSpy = function(putativeSpy) {
  return putativeSpy && putativeSpy.isSpy;
};

/**
 * Creates a more complicated spy: an Object that has every property a function that is a spy.  Used for stubbing something
 * large in one call.
 *
 * @param {String} baseName name of spy class
 * @param {Array} methodNames array of names of methods to make spies
 */
jasmine.createSpyObj = function(baseName, methodNames) {
  if (!jasmine.isArray_(methodNames) || methodNames.length === 0) {
    throw new Error('createSpyObj requires a non-empty array of method names to create spies for');
  }
  var obj = {};
  for (var i = 0; i < methodNames.length; i++) {
    obj[methodNames[i]] = jasmine.createSpy(baseName + '.' + methodNames[i]);
  }
  return obj;
};

/**
 * All parameters are pretty-printed and concatenated together, then written to the current spec's output.
 *
 * Be careful not to leave calls to <code>jasmine.log</code> in production code.
 */
jasmine.log = function() {
  var spec = jasmine.getEnv().currentSpec;
  spec.log.apply(spec, arguments);
};

/**
 * Function that installs a spy on an existing object's method name.  Used within a Spec to create a spy.
 *
 * @example
 * // spy example
 * var foo = {
 *   not: function(bool) { return !bool; }
 * }
 * spyOn(foo, 'not'); // actual foo.not will not be called, execution stops
 *
 * @see jasmine.createSpy
 * @param obj
 * @param methodName
 * @returns a Jasmine spy that can be chained with all spy methods
 */
var spyOn = function(obj, methodName) {
  return jasmine.getEnv().currentSpec.spyOn(obj, methodName);
};
if (isCommonJS) exports.spyOn = spyOn;


/**
 * Creates a Jasmine spec that will be added to the current suite.
 *
 * // TODO: pending tests
 *
 * @example
 * it('should be true', function() {
 *   expect(true).toEqual(true);
 * });
 *
 * @param {String} desc description of this specification
 * @param {Function} func defines the preconditions and expectations of the spec
 */
var it = function(desc, func) {
  return jasmine.getEnv().it(desc, func);
};
if (isCommonJS) exports.it = it;

/**
 * Creates a <em>disabled</em> Jasmine spec.
 *
 * A convenience method that allows existing specs to be disabled temporarily during development.
 *
 * @param {String} desc description of this specification
 * @param {Function} func defines the preconditions and expectations of the spec
 */
var xit = function(desc, func) {
  return jasmine.getEnv().xit(desc, func);
};
if (isCommonJS) exports.xit = xit;


/**
 * Starts a chain for a Jasmine expectation.
 *
 * It is passed an Object that is the actual value and should chain to one of the many
 * jasmine.Matchers functions.
 *
 * @param {Object} actual Actual value to test against and expected value
 */
var expect = function(actual) {
  return jasmine.getEnv().currentSpec.expect(actual);
};
if (isCommonJS) exports.expect = expect;

/**
 * Defines part of a jasmine spec.  Used in cominbination with waits or waitsFor in asynchrnous specs.
 *
 * @param {Function} func Function that defines part of a jasmine spec.
 */
var runs = function(func) {
  jasmine.getEnv().currentSpec.runs(func);
};
if (isCommonJS) exports.runs = runs;

/**
 * Waits a fixed time period before moving to the next block.
 *
 * @deprecated Use waitsFor() instead
 * @param {Number} timeout milliseconds to wait
 */
var waits = function(timeout) {
  jasmine.getEnv().currentSpec.waits(timeout);
};
if (isCommonJS) exports.waits = waits;

/**
 * Waits for the latchFunction to return true before proceeding to the next block.
 *
 * @param {Function} latchFunction
 * @param {String} optional_timeoutMessage
 * @param {Number} optional_timeout
 */
var waitsFor = function(latchFunction, optional_timeoutMessage,
optional_timeout) {
  jasmine.getEnv().currentSpec.waitsFor.apply(jasmine.getEnv().currentSpec,
arguments);
};
```

```
if (isCommonJS) exports.waitsFor = waitsFor;

/**
 * A function that is called before each spec in a suite.
 *
 * Used for spec setup, including validating assumptions.
 *
 * @param {Function} beforeEachFunction
 */
var beforeEach = function(beforeEachFunction) {
  jasmine.getEnv().beforeEach(beforeEachFunction);
};
if (isCommonJS) exports.beforeEach = beforeEach;

/**
 * A function that is called after each spec in a suite.
 *
 * Used for restoring any state that is hijacked during spec execution.
 *
 * @param {Function} afterEachFunction
 */
var afterEach = function(afterEachFunction) {
  jasmine.getEnv().afterEach(afterEachFunction);
};
if (isCommonJS) exports.afterEach = afterEach;

/**
 * Defines a suite of specifications.
 *
 * Stores the description and all defined specs in the Jasmine environment as one suite of specs. Variables declared
 * are accessible by calls to beforeEach, it, and afterEach. Describe blocks can be nested, allowing for specialization
 * of setup in some tests.
 *
 * @example
 * // TODO: a simple suite
 *
 * // TODO: a simple suite with a nested describe block
 *
 * @param {String} description A string, usually the class under test.
 * @param {Function} specDefinitions function that defines several specs.
 */
var describe = function(description, specDefinitions) {
  return jasmine.getEnv().describe(description, specDefinitions);
};
if (isCommonJS) exports.describe = describe;

/**
 * Disables a suite of specifications.  Used to disable some suites in a file, or files, temporarily during development.
 *
 * @param {String} description A string, usually the class under test.
 * @param {Function} specDefinitions function that defines several specs.
 */
var xdescribe = function(description, specDefinitions) {
  return jasmine.getEnv().xdescribe(description, specDefinitions);
};
if (isCommonJS) exports.xdescribe = xdescribe;
```

```
// Provide the XMLHttpRequest class for IE 5.x-6.x:
jasmine.XmlHttpRequest = (typeof XMLHttpRequest == "undefined") ? function() {
  function tryIt(f) {
    try {
      return f();
    } catch(e) {
    }
    return null;
  }

  var xhr = tryIt(function() {
    return new ActiveXObject("Msxml2.XMLHTTP.6.0");
  }) ||
    tryIt(function() {
      return new ActiveXObject("Msxml2.XMLHTTP.3.0");
    }) ||
    tryIt(function() {
      return new ActiveXObject("Msxml2.XMLHTTP");
    }) ||
    tryIt(function() {
      return new ActiveXObject("Microsoft.XMLHTTP");
    });

  if (!xhr) throw new Error("This browser does not support XMLHttpRequest.");

  return xhr;
} : XMLHttpRequest;
/**
 * @namespace
 */
jasmine.util = {};

/**
 * Declare that a child class inherit it's prototype from the parent class.
 *
 * @private
 * @param {Function} childClass
 * @param {Function} parentClass
 */
jasmine.util.inherit = function(childClass, parentClass) {
  /**
   * @private
   */
  var subclass = function() {
  };
  subclass.prototype = parentClass.prototype;
  childClass.prototype = new subclass();
};

jasmine.util.formatException = function(e) {
  var lineNumber;
  if (e.line) {
    lineNumber = e.line;
  }
  else if (e.lineNumber) {
   lineNumber = e.lineNumber;
  }

  var file;

  if (e.sourceURL) {
   file = e.sourceURL;
  }
  else if (e.fileName) {
   file = e.fileName;
  }

  var message = (e.name && e.message) ? (e.name + ': ' + e.message) : e.toString();

  if (file && lineNumber) {
   message += ' in ' + file + ' (line ' + lineNumber + ')';
  }

  return message;
```

```
};

jasmine.util.htmlEscape = function(str) {
  if (!str) return str;
  return str.replace(/&/g, '&amp;')
    .replace(/</g, '&lt;')
    .replace(/>/g, '&gt;');
};

jasmine.util.argsToArray = function(args) {
  var arrayOfArgs = [];
  for (var i = 0; i < args.length; i++) arrayOfArgs.push(args[i]);
  return arrayOfArgs;
};

jasmine.util.extend = function(destination, source) {
  for (var property in source) destination[property] = source[property];
  return destination;
};
```

```
/**
 * Environment for Jasmine
 *
 * @constructor
 */
jasmine.Env = function() {
  this.currentSpec = null;
  this.currentSuite = null;
  this.currentRunner_ = new jasmine.Runner(this);

  this.reporter = new jasmine.MultiReporter();

  this.updateInterval = jasmine.DEFAULT_UPDATE_INTERVAL;
  this.defaultTimeoutInterval = jasmine.DEFAULT_TIMEOUT_INTERVAL;
  this.lastUpdate = 0;
  this.specFilter = function() {
    return true;
  };
  this.nextSpecId_ = 0;

  this.nextSuiteId_ = 0;

  this.equalityTesters_ = [];


  // wrap matchers

  this.matchersClass = function() {

    jasmine.Matchers.apply(this, arguments);

  };

  jasmine.util.inherit(this.matchersClass, jasmine.Matchers);


  jasmine.Matchers.wrapInto_(jasmine.Matchers.prototype, this.matchersClass);

};




jasmine.Env.prototype.setTimeout = jasmine.setTimeout;

jasmine.Env.prototype.clearTimeout = jasmine.clearTimeout;

jasmine.Env.prototype.setInterval = jasmine.setInterval;

jasmine.Env.prototype.clearInterval = jasmine.clearInterval;
```

```
/**
 * @returns an object containing jasmine version build info, if set.
 */
jasmine.Env.prototype.version = function () {
  if (jasmine.version_) {
    return jasmine.version_;
  } else {
    throw new Error('Version not set');
  }
};

/**
 * @returns string containing jasmine version build info, if set.
 */
jasmine.Env.prototype.versionString = function() {
  if (!jasmine.version_) {
    return "version unknown";
  }
  var version = this.version();
  var versionString = version.major + "." + version.minor + "." + version.build;
  if (version.release_candidate) {
    versionString += ".rc" + version.release_candidate;
  }
  versionString += " revision " + version.revision;
  return versionString;
};

/**
 * @returns a sequential integer starting at 0
 */
jasmine.Env.prototype.nextSpecId = function () {
  return this.nextSpecId_++;
};

/**
 * @returns a sequential integer starting at 0
 */
jasmine.Env.prototype.nextSuiteId = function () {
  return this.nextSuiteId_++;
};

/**
 * Register a reporter to receive status updates from Jasmine.
 * @param {jasmine.Reporter} reporter An object which will receive status updates.
 */
jasmine.Env.prototype.addReporter = function(reporter) {
  this.reporter.addReporter(reporter);
};

jasmine.Env.prototype.execute = function() {
  this.currentRunner_.execute();
};

jasmine.Env.prototype.describe = function(description, specDefinitions) {
  var suite = new jasmine.Suite(this, description, specDefinitions, this.currentSuite);

  var parentSuite = this.currentSuite;
  if (parentSuite) {
    parentSuite.add(suite);
  } else {
    this.currentRunner_.add(suite);
  }

  this.currentSuite = suite;

  var declarationError = null;
  try {
    specDefinitions.call(suite);
  } catch(e) {
    declarationError = e;
  }

  if (declarationError) {
    this.it("encountered a declaration exception", function() {
      throw declarationError;
    });
  }

  this.currentSuite = parentSuite;

  return suite;
};

jasmine.Env.prototype.beforeEach = function(beforeEachFunction) {
  if (this.currentSuite) {
    this.currentSuite.beforeEach(beforeEachFunction);
  } else {
    this.currentRunner_.beforeEach(beforeEachFunction);
  }
};

jasmine.Env.prototype.currentRunner = function () {
```

```
  return this.currentRunner_;
};

jasmine.Env.prototype.afterEach = function(afterEachFunction) {
  if (this.currentSuite) {
    this.currentSuite.afterEach(afterEachFunction);
  } else {
    this.currentRunner_.afterEach(afterEachFunction);
  }

};

jasmine.Env.prototype.xdescribe = function(desc, specDefinitions) {
  return {
    execute: function() {
    }
  };
};
```

```
jasmine.Env.prototype.it = function(description, func) {
  var spec = new jasmine.Spec(this, this.currentSuite, description);
  this.currentSuite.add(spec);
  this.currentSpec = spec;

  if (func) {
    spec.runs(func);
  }

  return spec;
};

jasmine.Env.prototype.xit = function(desc, func) {
  return {
    id: this.nextSpecId(),
    runs: function() {
    }
  };
};
```

```
jasmine.Env.prototype.compareObjects_ = function(a, b, mismatchKeys,
mismatchValues) {
 if (a.__Jasmine_been_here_before__ === b &&
b.__Jasmine_been_here_before__ === a) {
  return true;
 }

 a.__Jasmine_been_here_before__ = b;
 b.__Jasmine_been_here_before__ = a;

 var hasKey = function(obj, keyName) {
  return obj !== null && obj[keyName] !== jasmine.undefined;
 };

 for (var property in b) {
  if (!hasKey(a, property) && hasKey(b, property)) {
   mismatchKeys.push("expected has key '" + property + "', but missing from
actual.");
  }
  }
  for (property in a) {
    if (!hasKey(b, property) && hasKey(a, property)) {
      mismatchKeys.push("expected missing key '" + property + "', but present in
actual.");
    }
  }
  for (property in b) {
    if (property == '__Jasmine_been_here_before__') continue;
    if (!this.equals_(a[property], b[property], mismatchKeys, mismatchValues)) {
      mismatchValues.push("'" + property + "' was '" + (b[property] ?
jasmine.util.htmlEscape(b[property].toString()) : b[property]) + "' in expected, but
was '" + (a[property] ? jasmine.util.htmlEscape(a[property].toString()) :
a[property]) + "' in actual.");
    }
  }

  if (jasmine.isArray_(a) && jasmine.isArray_(b) && a.length != b.length) {
    mismatchValues.push("arrays were not the same length");
  }
  delete a.__Jasmine_been_here_before__;
  delete b.__Jasmine_been_here_before__;
  return (mismatchKeys.length === 0 && mismatchValues.length === 0);
};

jasmine.Env.prototype.equals_ = function(a, b, mismatchKeys, mismatchValues)
{
  mismatchKeys = mismatchKeys || [];
  mismatchValues = mismatchValues || [];

  for (var i = 0; i < this.equalityTesters_.length; i++) {
    var equalityTester = this.equalityTesters_[i];
    var result = equalityTester(a, b, this, mismatchKeys, mismatchValues);
    if (result !== jasmine.undefined) return result;
  }

  if (a === b) return true;

  if (a === jasmine.undefined || a === null || b === jasmine.undefined || b ===
null) {
  return (a == jasmine.undefined && b == jasmine.undefined);

}


if (jasmine.isDomNode(a) && jasmine.isDomNode(b)) {

  return a === b;

}


if (a instanceof Date && b instanceof Date) {

  return a.getTime() == b.getTime();

}


if (a.jasmineMatches) {

  return a.jasmineMatches(b);

}


if (b.jasmineMatches) {

  return b.jasmineMatches(a);

}
```

```
if (a instanceof jasmine.Matchers.ObjectContaining) {
  return a.matches(b);
}

if (b instanceof jasmine.Matchers.ObjectContaining) {
  return b.matches(a);
}

if (jasmine.isString_(a) && jasmine.isString_(b)) {
  return (a == b);
}

if (jasmine.isNumber_(a) && jasmine.isNumber_(b)) {
  return (a == b);
}

if (typeof a === "object" && typeof b === "object") {
  return this.compareObjects_(a, b, mismatchKeys, mismatchValues);
}
```

```
  //Straight check
  return (a === b);
};

jasmine.Env.prototype.contains_ = function(haystack, needle) {
  if (jasmine.isArray_(haystack)) {
    for (var i = 0; i < haystack.length; i++) {
      if (this.equals_(haystack[i], needle)) return true;
    }
    return false;
  }
  return haystack.indexOf(needle) >= 0;
};

jasmine.Env.prototype.addEqualityTester = function(equalityTester) {
  this.equalityTesters_.push(equalityTester);
};
/** No-op base class for Jasmine reporters.
```

```
 *
 * @constructor
 */
jasmine.Reporter = function() {
};

//noinspection JSUnusedLocalSymbols
jasmine.Reporter.prototype.reportRunnerStarting = function(runner) {
};

//noinspection JSUnusedLocalSymbols
jasmine.Reporter.prototype.reportRunnerResults = function(runner) {
};

//noinspection JSUnusedLocalSymbols
jasmine.Reporter.prototype.reportSuiteResults = function(suite) {
};

//noinspection JSUnusedLocalSymbols
```

```
jasmine.Reporter.prototype.reportSpecStarting = function(spec) {
};

//noinspection JSUnusedLocalSymbols
jasmine.Reporter.prototype.reportSpecResults = function(spec) {
};

//noinspection JSUnusedLocalSymbols
jasmine.Reporter.prototype.log = function(str) {
};

/**
 * Blocks are functions with executable code that make up a spec.
 *
 * @constructor
 * @param {jasmine.Env} env
 * @param {Function} func
 * @param {jasmine.Spec} spec
 */
```

```
jasmine.Block = function(env, func, spec) {
  this.env = env;
  this.func = func;
  this.spec = spec;
};


jasmine.Block.prototype.execute = function(onComplete) {
  try {
    this.func.apply(this.spec);
  } catch (e) {
    this.spec.fail(e);
  }
  onComplete();
};
/** JavaScript API reporter.
 *
 * @constructor
 */
jasmine.JsApiReporter = function() {
  this.started = false;

  this.finished = false;

  this.suites_ = [];

  this.results_ = {};

};


jasmine.JsApiReporter.prototype.reportRunnerStarting = function(runner) {

  this.started = true;

  var suites = runner.topLevelSuites();

  for (var i = 0; i < suites.length; i++) {

    var suite = suites[i];

    this.suites_.push(this.summarize_(suite));

  }

};


jasmine.JsApiReporter.prototype.suites = function() {

  return this.suites_;

};
```

```
jasmine.JsApiReporter.prototype.summarize_ = function(suiteOrSpec) {
  var isSuite = suiteOrSpec instanceof jasmine.Suite;
  var summary = {
    id: suiteOrSpec.id,
    name: suiteOrSpec.description,
    type: isSuite ? 'suite' : 'spec',
    children: []
  };

  if (isSuite) {
    var children = suiteOrSpec.children();
    for (var i = 0; i < children.length; i++) {
      summary.children.push(this.summarize_(children[i]));
    }
  }
  return summary;
};

jasmine.JsApiReporter.prototype.results = function() {
  return this.results_;
};

jasmine.JsApiReporter.prototype.resultsForSpec = function(specId) {
  return this.results_[specId];
};

//noinspection JSUnusedLocalSymbols
jasmine.JsApiReporter.prototype.reportRunnerResults = function(runner) {
  this.finished = true;
};

//noinspection JSUnusedLocalSymbols
jasmine.JsApiReporter.prototype.reportSuiteResults = function(suite) {
};

//noinspection JSUnusedLocalSymbols
jasmine.JsApiReporter.prototype.reportSpecResults = function(spec) {
  this.results_[spec.id] = {
    messages: spec.results().getItems(),
    result: spec.results().failedCount > 0 ? "failed" : "passed"
  };
};

//noinspection JSUnusedLocalSymbols
jasmine.JsApiReporter.prototype.log = function(str) {
};

jasmine.JsApiReporter.prototype.resultsForSpecs = function(specIds){
  var results = {};
  for (var i = 0; i < specIds.length; i++) {
    var specId = specIds[i];
    results[specId] = this.summarizeResult_(this.results_[specId]);
  }
  return results;
};

jasmine.JsApiReporter.prototype.summarizeResult_ = function(result){
  var summaryMessages = [];
  var messagesLength = result.messages.length;
  for (var messageIndex = 0; messageIndex < messagesLength; messageIndex++)
{
    var resultMessage = result.messages[messageIndex];
    summaryMessages.push({
      text: resultMessage.type == 'log' ? resultMessage.toString() :
jasmine.undefined,
      passed: resultMessage.passed ? resultMessage.passed() : true,
      type: resultMessage.type,
      message: resultMessage.message,
      trace: {
        stack: resultMessage.passed && !resultMessage.passed() ?
resultMessage.trace.stack : jasmine.undefined
      }
    });
  }

  return {
    result : result.result,
    messages : summaryMessages
  };
};

/**
 * @constructor
 * @param {jasmine.Env} env
 * @param actual
 * @param {jasmine.Spec} spec
 */
jasmine.Matchers = function(env, actual, spec, opt_isNot) {
  this.env = env;
  this.actual = actual;
  this.spec = spec;
  this.isNot = opt_isNot || false;
  this.reportWasCalled_ = false;
};

// todo: @deprecated as of Jasmine 0.11, remove soon [xw]
```

```
jasmine.Matchers.pp = function(str) {
  throw new Error("jasmine.Matchers.pp() is no longer supported, please use jasmine.pp() instead!");
};

// todo: @deprecated Deprecated as of Jasmine 0.10. Rewrite your custom matchers to return true or false. [xw]
jasmine.Matchers.prototype.report = function(result, failing_message, details) {
  throw new Error("As of jasmine 0.11, custom matchers must be implemented differently -- please see jasmine docs");
};

jasmine.Matchers.wrapInto_ = function(prototype, matchersClass) {
  for (var methodName in prototype) {
    if (methodName == 'report') continue;
    var orig = prototype[methodName];
    matchersClass.prototype[methodName] = jasmine.Matchers.matcherFn_(methodName, orig);
  }
};
```

```
jasmine.Matchers.matcherFn_ = function(matcherName, matcherFunction) {

  return function() {

    var matcherArgs = jasmine.util.argsToArray(arguments);

    var result = matcherFunction.apply(this, arguments);



    if (this.isNot) {

      result = !result;

    }



    if (this.reportWasCalled_) return result;



    var message;

    if (!result) {

      if (this.message) {

        message = this.message.apply(this, arguments);

        if (jasmine.isArray_(message)) {

          message = message[this.isNot ? 1 : 0];

        }

      } else {
        var englishyPredicate = matcherName.replace(/[A-Z]/g, function(s) { return '
' + s.toLowerCase(); });
        message = "Expected " + jasmine.pp(this.actual) + (this.isNot ? " not " : " ")
+ englishyPredicate;
        if (matcherArgs.length > 0) {
          for (var i = 0; i < matcherArgs.length; i++) {
            if (i > 0) message += ",";
            message += " " + jasmine.pp(matcherArgs[i]);
          }
        }
        message += ".";
      }
    }
    var expectationResult = new jasmine.ExpectationResult({
      matcherName: matcherName,
      passed: result,
      expected: matcherArgs.length > 1 ? matcherArgs : matcherArgs[0],
      actual: this.actual,
      message: message
    });
    this.spec.addMatcherResult(expectationResult);
    return jasmine.undefined;
  };
};




/**
 * toBe: compares the actual to the expected using ===
 * @param expected
 */
jasmine.Matchers.prototype.toBe = function(expected) {
  return this.actual === expected;
};

/**
 * toNotBe: compares the actual to the expected using !==
 * @param expected
 * @deprecated as of 1.0. Use not.toBe() instead.
 */
jasmine.Matchers.prototype.toNotBe = function(expected) {
  return this.actual !== expected;
};

/**
 * toEqual: compares the actual to the expected using common sense equality. Handles Objects, Arrays, etc.
 *
 * @param expected
 */
jasmine.Matchers.prototype.toEqual = function(expected) {
  return this.env.equals_(this.actual, expected);
};

/**
 * toNotEqual: compares the actual to the expected using the ! of jasmine.Matchers.toEqual
 * @param expected
 * @deprecated as of 1.0. Use not.toEqual() instead.
 */
jasmine.Matchers.prototype.toNotEqual = function(expected) {
  return !this.env.equals_(this.actual, expected);
};

/**
 * Matcher that compares the actual to the expected using a regular expression. Constructs a RegExp, so takes
 * a pattern or a String.
 *
 * @param expected
 */
jasmine.Matchers.prototype.toMatch = function(expected) {
  return new RegExp(expected).test(this.actual);
};

/**
 * Matcher that compares the actual to the expected using the boolean inverse of jasmine.Matchers.toMatch
 * @param expected
 * @deprecated as of 1.0. Use not.toMatch() instead.
 */
jasmine.Matchers.prototype.toNotMatch = function(expected) {
  return !(new RegExp(expected).test(this.actual));
};

/**
 * Matcher that compares the actual to jasmine.undefined.
 */
jasmine.Matchers.prototype.toBeDefined = function() {
  return (this.actual !== jasmine.undefined);
};

/**
 * Matcher that compares the actual to jasmine.undefined.
 */
jasmine.Matchers.prototype.toBeUndefined = function() {
  return (this.actual === jasmine.undefined);
};

/**
 * Matcher that compares the actual to null.
 */
jasmine.Matchers.prototype.toBeNull = function() {
  return (this.actual === null);
};

/**
 * Matcher that boolean not-nots the actual.
 */
jasmine.Matchers.prototype.toBeTruthy = function() {
  return !!this.actual;
};


/**
 * Matcher that boolean nots the actual.
 */
jasmine.Matchers.prototype.toBeFalsy = function() {
  return !this.actual;
};


/**
 * Matcher that checks to see if the actual, a Jasmine spy, was called.
 */
jasmine.Matchers.prototype.toHaveBeenCalled = function() {
  if (arguments.length > 0) {
    throw new Error('toHaveBeenCalled does not take arguments, use toHaveBeenCalledWith');
  }

  if (!jasmine.isSpy(this.actual)) {
    throw new Error('Expected a spy, but got ' + jasmine.pp(this.actual) + '.');
  }

  this.message = function() {
    return [
      "Expected spy " + this.actual.identity + " to have been called.",
      "Expected spy " + this.actual.identity + " not to have been called."
    ];
  };

  return this.actual.wasCalled;
};

/** @deprecated Use expect(xxx).toHaveBeenCalled() instead */
jasmine.Matchers.prototype.wasCalled =
jasmine.Matchers.prototype.toHaveBeenCalled;

/**
 * Matcher that checks to see if the actual, a Jasmine spy, was not called.
 *
 * @deprecated Use expect(xxx).not.toHaveBeenCalled() instead
 */
jasmine.Matchers.prototype.wasNotCalled = function() {
  if (arguments.length > 0) {
    throw new Error('wasNotCalled does not take arguments');
  }

  if (!jasmine.isSpy(this.actual)) {
    throw new Error('Expected a spy, but got ' + jasmine.pp(this.actual) + '.');
  }

  this.message = function() {
    return [
      "Expected spy " + this.actual.identity + " to not have been called.",
      "Expected spy " + this.actual.identity + " to have been called."
    ];
  };

  return !this.actual.wasCalled;
};

/**
 * Matcher that checks to see if the actual, a Jasmine spy, was called with a set of parameters.
 *
 * @example
 *
 */
jasmine.Matchers.prototype.toHaveBeenCalledWith = function() {
  var expectedArgs = jasmine.util.argsToArray(arguments);
  if (!jasmine.isSpy(this.actual)) {
    throw new Error('Expected a spy, but got ' + jasmine.pp(this.actual) + '.');
  }
  this.message = function() {
    if (this.actual.callCount === 0) {
      // todo: what should the failure message for .not.toHaveBeenCalledWith() be? is this right? test better. [xw]
      return [
        "Expected spy " + this.actual.identity + " to have been called with " + jasmine.pp(expectedArgs) + " but it was never called.",
        "Expected spy " + this.actual.identity + " not to have been called with " + jasmine.pp(expectedArgs) + " but it was."
      ];
    } else {
      return [
        "Expected spy " + this.actual.identity + " to have been called with " +
jasmine.pp(expectedArgs) + " but was called with " +
jasmine.pp(this.actual.argsForCall),
        "Expected spy " + this.actual.identity + " not to have been called with " +
jasmine.pp(expectedArgs) + " but was called with " +
jasmine.pp(this.actual.argsForCall)
      ];
    }
  };

  return this.env.contains_(this.actual.argsForCall, expectedArgs);
};

/** @deprecated Use expect(xxx).toHaveBeenCalledWith() instead */
jasmine.Matchers.prototype.wasCalledWith =
jasmine.Matchers.prototype.toHaveBeenCalledWith;

/** @deprecated Use expect(xxx).not.toHaveBeenCalledWith() instead */
jasmine.Matchers.prototype.wasNotCalledWith = function() {
  var expectedArgs = jasmine.util.argsToArray(arguments);
  if (!jasmine.isSpy(this.actual)) {
    throw new Error('Expected a spy, but got ' + jasmine.pp(this.actual) + '.');
  }

  this.message = function() {
    return [
      "Expected spy not to have been called with " + jasmine.pp(expectedArgs) + " but it was",
      "Expected spy to have been called with " + jasmine.pp(expectedArgs) + " but it was"
    ];
  };

  return !this.env.contains_(this.actual.argsForCall, expectedArgs);
};

/**
 * Matcher that checks that the expected item is an element in the actual Array.
 *
 * @param {Object} expected
 */
jasmine.Matchers.prototype.toContain = function(expected) {
  return this.env.contains_(this.actual, expected);
};

/**
 * Matcher that checks that the expected item is NOT an element in the actual Array.
 *
 * @param {Object} expected
 * @deprecated as of 1.0. Use not.toContain() instead.
 */
jasmine.Matchers.prototype.toNotContain = function(expected) {
  return !this.env.contains_(this.actual, expected);
};

jasmine.Matchers.prototype.toBeLessThan = function(expected) {
  return this.actual < expected;
};
```

```
jasmine.Matchers.prototype.toBeGreaterThan = function(expected) {
  return this.actual > expected;
};

/**
 * Matcher that checks that the expected item is equal to the actual item
 * up to a given level of decimal precision (default 2).
 *
 * @param {Number} expected
 * @param {Number} precision
 */
jasmine.Matchers.prototype.toBeCloseTo = function(expected, precision) {
  if (!(precision === 0)) {
    precision = precision || 2;
  }
  var multiplier = Math.pow(10, precision);
  var actual = Math.round(this.actual * multiplier);
  expected = Math.round(expected * multiplier);
  return expected == actual;
```

```
};


/**
 * Matcher that checks that the expected exception was thrown by the actual.
 *
 * @param {String} expected
 */
jasmine.Matchers.prototype.toThrow = function(expected) {
  var result = false;
  var exception;
  if (typeof this.actual != 'function') {
    throw new Error('Actual is not a function');
  }
  try {
    this.actual();
  } catch (e) {
    exception = e;
  }
  if (exception) {
    result = (expected === jasmine.undefined || this.env.equals_(exception.message || exception, expected.message || expected));
  }

  var not = this.isNot ? "not " : "";

  this.message = function() {
    if (exception && (expected === jasmine.undefined || !this.env.equals_(exception.message || exception, expected.message || expected))) {
      return ["Expected function " + not + "to throw", expected ? expected.message || expected : "an exception", ", but it threw", exception.message || exception].join(' ');
    } else {
      return "Expected function to throw an exception.";
    }
  };

  return result;
};
```

```
jasmine.Matchers.Any = function(expectedClass) {
  this.expectedClass = expectedClass;
};

jasmine.Matchers.Any.prototype.jasmineMatches = function(other) {
  if (this.expectedClass == String) {
    return typeof other == 'string' || other instanceof String;
  }

  if (this.expectedClass == Number) {
    return typeof other == 'number' || other instanceof Number;
  }

  if (this.expectedClass == Function) {
    return typeof other == 'function' || other instanceof Function;
  }

  if (this.expectedClass == Object) {
    return typeof other == 'object';
  }

  return other instanceof this.expectedClass;
};

jasmine.Matchers.Any.prototype.jasmineToString = function() {
  return '<jasmine.any(' + this.expectedClass + ')>';
};

jasmine.Matchers.ObjectContaining = function (sample) {
  this.sample = sample;
};

jasmine.Matchers.ObjectContaining.prototype.jasmineMatches = function(other,
mismatchKeys, mismatchValues) {
  mismatchKeys = mismatchKeys || [];
  mismatchValues = mismatchValues || [];

  var env = jasmine.getEnv();
  var hasKey = function(obj, keyName) {
    return obj != null && obj[keyName] !== jasmine.undefined;
  };

  for (var property in this.sample) {
    if (!hasKey(other, property) && hasKey(this.sample, property)) {
      mismatchKeys.push("expected has key '" + property + "', but missing from actual.");
    }
    else if (!env.equals_(this.sample[property], other[property], mismatchKeys, mismatchValues)) {
      mismatchValues.push("'" + property + "' was '" + (other[property] ? jasmine.util.htmlEscape(other[property].toString()) : other[property]) + "' in expected, but was '" + (this.sample[property] ? jasmine.util.htmlEscape(this.sample[property].toString()) : this.sample[property]) + "' in actual.");
    }
  }

  return (mismatchKeys.length === 0 && mismatchValues.length === 0);
};
```

```
jasmine.Matchers.ObjectContaining.prototype.jasmineToString = function () {
  return "<jasmine.objectContaining(" + jasmine.pp(this.sample) + ")>";
};
// Mock setTimeout, clearTimeout
// Contributed by Pivotal Computer Systems, www.pivotalsf.com

jasmine.FakeTimer = function() {
  this.reset();

  var self = this;
  self.setTimeout = function(funcToCall, millis) {
    self.timeoutsMade++;
    self.scheduleFunction(self.timeoutsMade, funcToCall, millis, false);
    return self.timeoutsMade;
  };

  self.setInterval = function(funcToCall, millis) {
    self.timeoutsMade++;
    self.scheduleFunction(self.timeoutsMade, funcToCall, millis, true);

    return self.timeoutsMade;

  };


  self.clearTimeout = function(timeoutKey) {

    self.scheduledFunctions[timeoutKey] = jasmine.undefined;

  };


  self.clearInterval = function(timeoutKey) {

    self.scheduledFunctions[timeoutKey] = jasmine.undefined;

  };


};


jasmine.FakeTimer.prototype.reset = function() {

  this.timeoutsMade = 0;

  this.scheduledFunctions = {};

  this.nowMillis = 0;

};
```

```
jasmine.FakeTimer.prototype.tick = function(millis) {
  var oldMillis = this.nowMillis;
  var newMillis = oldMillis + millis;
  this.runFunctionsWithinRange(oldMillis, newMillis);
  this.nowMillis = newMillis;
};

jasmine.FakeTimer.prototype.runFunctionsWithinRange = function(oldMillis, nowMillis) {
  var scheduledFunc;
  var funcsToRun = [];
  for (var timeoutKey in this.scheduledFunctions) {
    scheduledFunc = this.scheduledFunctions[timeoutKey];
    if (scheduledFunc != jasmine.undefined &&
        scheduledFunc.runAtMillis >= oldMillis &&
        scheduledFunc.runAtMillis <= nowMillis) {
      funcsToRun.push(scheduledFunc);
      this.scheduledFunctions[timeoutKey] = jasmine.undefined;
    }
```

```
}

if (funcsToRun.length > 0) {
  funcsToRun.sort(function(a, b) {
    return a.runAtMillis - b.runAtMillis;
  });
  for (var i = 0; i < funcsToRun.length; ++i) {
    try {
      var funcToRun = funcsToRun[i];
      this.nowMillis = funcToRun.runAtMillis;
      funcToRun.funcToCall();
      if (funcToRun.recurring) {
        this.scheduleFunction(funcToRun.timeoutKey,
            funcToRun.funcToCall,
            funcToRun.millis,
            true);
      }
    } catch(e) {
    }
    }
    this.runFunctionsWithinRange(oldMillis, nowMillis);
  }
};

jasmine.FakeTimer.prototype.scheduleFunction         =         function(timeoutKey,
funcToCall, millis, recurring) {
  this.scheduledFunctions[timeoutKey] = {
    runAtMillis: this.nowMillis + millis,
    funcToCall: funcToCall,
    recurring: recurring,
    timeoutKey: timeoutKey,
    millis: millis
  };
};

/**
 * @namespace
 */
jasmine.Clock = {
  defaultFakeTimer: new jasmine.FakeTimer(),

  reset: function() {
    jasmine.Clock.assertInstalled();
    jasmine.Clock.defaultFakeTimer.reset();
  },

  tick: function(millis) {
    jasmine.Clock.assertInstalled();
    jasmine.Clock.defaultFakeTimer.tick(millis);
  },

  runFunctionsWithinRange: function(oldMillis, nowMillis) {
    jasmine.Clock.defaultFakeTimer.runFunctionsWithinRange(oldMillis,
nowMillis);
  },

  scheduleFunction: function(timeoutKey, funcToCall, millis, recurring) {
    jasmine.Clock.defaultFakeTimer.scheduleFunction(timeoutKey, funcToCall,
millis, recurring);
  },

  useMock: function() {
    if (!jasmine.Clock.isInstalled()) {
      var spec = jasmine.getEnv().currentSpec;
      spec.after(jasmine.Clock.uninstallMock);

      jasmine.Clock.installMock();
    }
  },

  installMock: function() {
    jasmine.Clock.installed = jasmine.Clock.defaultFakeTimer;
  },

  uninstallMock: function() {
    jasmine.Clock.assertInstalled();
    jasmine.Clock.installed = jasmine.Clock.real;
  },
  real: {
    setTimeout: jasmine.getGlobal().setTimeout,
    clearTimeout: jasmine.getGlobal().clearTimeout,
    setInterval: jasmine.getGlobal().setInterval,
    clearInterval: jasmine.getGlobal().clearInterval
  },

  assertInstalled: function() {
    if (!jasmine.Clock.isInstalled()) {
      throw new Error("Mock clock is not installed, use jasmine.Clock.useMock()");
    }
  },

  isInstalled: function() {
    return jasmine.Clock.installed == jasmine.Clock.defaultFakeTimer;
  },

  installed: null
```

```
};
jasmine.Clock.installed = jasmine.Clock.real;


//else for IE support
jasmine.getGlobal().setTimeout = function(funcToCall, millis) {
  if (jasmine.Clock.installed.setTimeout.apply) {
    return jasmine.Clock.installed.setTimeout.apply(this, arguments);
  } else {
    return jasmine.Clock.installed.setTimeout(funcToCall, millis);
  }
};


jasmine.getGlobal().setInterval = function(funcToCall, millis) {
  if (jasmine.Clock.installed.setInterval.apply) {
    return jasmine.Clock.installed.setInterval.apply(this, arguments);
  } else {
    return jasmine.Clock.installed.setInterval(funcToCall, millis);
  }
};
```

```
jasmine.getGlobal().clearTimeout = function(timeoutKey) {
  if (jasmine.Clock.installed.clearTimeout.apply) {
    return jasmine.Clock.installed.clearTimeout.apply(this, arguments);
  } else {
    return jasmine.Clock.installed.clearTimeout(timeoutKey);
  }
};

jasmine.getGlobal().clearInterval = function(timeoutKey) {
  if (jasmine.Clock.installed.clearTimeout.apply) {
    return jasmine.Clock.installed.clearInterval.apply(this, arguments);
  } else {
    return jasmine.Clock.installed.clearInterval(timeoutKey);
  }
};

/**
 * @constructor
 */
jasmine.MultiReporter = function() {
  this.subReporters_ = [];
};
jasmine.util.inherit(jasmine.MultiReporter, jasmine.Reporter);

jasmine.MultiReporter.prototype.addReporter = function(reporter) {
  this.subReporters_.push(reporter);
};

(function() {
  var functionNames = [
    "reportRunnerStarting",
    "reportRunnerResults",
    "reportSuiteResults",
    "reportSpecStarting",
    "reportSpecResults",
    "log"
  ];
  for (var i = 0; i < functionNames.length; i++) {
    var functionName = functionNames[i];
    jasmine.MultiReporter.prototype[functionName] = (function(functionName) {
      return function() {
        for (var j = 0; j < this.subReporters_.length; j++) {
          var subReporter = this.subReporters_[j];
          if (subReporter[functionName]) {
            subReporter[functionName].apply(subReporter, arguments);
          }
        }
      };
    })(functionName);
  }
})();
/**
 * Holds results for a set of Jasmine spec. Allows for the results array to hold another jasmine.NestedResults
 *
 * @constructor
 */
```

```
jasmine.NestedResults = function() {
  /**
   * The total count of results
   */
  this.totalCount = 0;
  /**
   * Number of passed results
   */
  this.passedCount = 0;
  /**
   * Number of failed results
   */
  this.failedCount = 0;
  /**
   * Was this suite/spec skipped?
   */
  this.skipped = false;
  /**
   * @ignore
   */
  this.items_ = [];
};

/**
 * Roll up the result counts.
 *
 * @param result
 */
jasmine.NestedResults.prototype.rollupCounts = function(result) {
  this.totalCount += result.totalCount;
  this.passedCount += result.passedCount;
  this.failedCount += result.failedCount;
};

/**
 * Adds a log message.
 * @param values Array of message parts which will be concatenated later.
 */
```

```
jasmine.NestedResults.prototype.log = function(values) {
  this.items_.push(new jasmine.MessageResult(values));
};

/**
 * Getter for the results: message & results.
 */
jasmine.NestedResults.prototype.getItems = function() {
  return this.items_;
};

/**
 * Adds a result, tracking counts (total, passed, & failed)
 * @param {jasmine.ExpectationResult|jasmine.NestedResults} result
 */
jasmine.NestedResults.prototype.addResult = function(result) {
  if (result.type != 'log') {
    if (result.items_) {
      this.rollupCounts(result);
    } else {
      this.totalCount++;
      if (result.passed()) {
        this.passedCount++;
      } else {
        this.failedCount++;
      }
    }
  }
  this.items_.push(result);
};

/**
 * @returns {Boolean} True if <b>everything</b> below passed
 */
jasmine.NestedResults.prototype.passed = function() {
  return this.passedCount === this.totalCount;
};
/**
```

```
 * Base class for pretty printing for expectation results.
 */
jasmine.PrettyPrinter = function() {
  this.ppNestLevel_ = 0;
};

/**
 * Formats a value in a nice, human-readable string.
 *
 * @param value
 */
jasmine.PrettyPrinter.prototype.format = function(value) {
  if (this.ppNestLevel_ > 40) {
    throw new Error('jasmine.PrettyPrinter: format() nested too deeply!');
  }

  this.ppNestLevel_++;
  try {
    if (value === jasmine.undefined) {
```

```
  this.emitScalar('undefined');

} else if (value === null) {

  this.emitScalar('null');

} else if (value === jasmine.getGlobal()) {

  this.emitScalar('<global>');

} else if (value.jasmineToString) {

  this.emitScalar(value.jasmineToString());

} else if (typeof value === 'string') {

  this.emitString(value);

} else if (jasmine.isSpy(value)) {

  this.emitScalar("spy on " + value.identity);

} else if (value instanceof RegExp) {

  this.emitScalar(value.toString());

} else if (typeof value === 'function') {

  this.emitScalar('Function');

} else if (typeof value.nodeType === 'number') {

  this.emitScalar('HTMLNode');

} else if (value instanceof Date) {

  this.emitScalar('Date(' + value + ')');
```

```
    } else if (value.__Jasmine_been_here_before__) {
      this.emitScalar('<circular reference: ' + (jasmine.isArray_(value) ? 'Array' : 'Object') + '>');
    } else if (jasmine.isArray_(value) || typeof value == 'object') {
      value.__Jasmine_been_here_before__ = true;
      if (jasmine.isArray_(value)) {
        this.emitArray(value);
      } else {
        this.emitObject(value);
      }
      delete value.__Jasmine_been_here_before__;
    } else {
      this.emitScalar(value.toString());
    }
  } finally {
    this.ppNestLevel_--;
  }
};

jasmine.PrettyPrinter.prototype.iterateObject = function(obj, fn) {
  for (var property in obj) {
    if (property == '__Jasmine_been_here_before__') continue;
    fn(property, obj.__lookupGetter__ ? (obj.__lookupGetter__(property) !==
jasmine.undefined &&
                                          obj.__lookupGetter__(property) !== null) : false);
  }
};

jasmine.PrettyPrinter.prototype.emitArray = jasmine.unimplementedMethod_;
jasmine.PrettyPrinter.prototype.emitObject = jasmine.unimplementedMethod_;
jasmine.PrettyPrinter.prototype.emitScalar = jasmine.unimplementedMethod_;
jasmine.PrettyPrinter.prototype.emitString = jasmine.unimplementedMethod_;

jasmine.StringPrettyPrinter = function() {
  jasmine.PrettyPrinter.call(this);

  this.string = '';
};
jasmine.util.inherit(jasmine.StringPrettyPrinter, jasmine.PrettyPrinter);
jasmine.StringPrettyPrinter.prototype.emitScalar = function(value) {

  this.append(value);

};


jasmine.StringPrettyPrinter.prototype.emitString = function(value) {

  this.append("'" + value + "'");

};


jasmine.StringPrettyPrinter.prototype.emitArray = function(array) {

  this.append('[ ');

  for (var i = 0; i < array.length; i++) {

    if (i > 0) {

      this.append(', ');

    }

    this.format(array[i]);

  }

  this.append(' ]');

};
```

```
jasmine.StringPrettyPrinter.prototype.emitObject = function(obj) {
  var self = this;
  this.append('{ ');
  var first = true;

  this.iterateObject(obj, function(property, isGetter) {
    if (first) {
      first = false;
    } else {
      self.append(', ');
    }

    self.append(property);
    self.append(' : ');
    if (isGetter) {
      self.append('<getter>');
    } else {
      self.format(obj[property]);
    }
  });

  this.append(' }');
};

jasmine.StringPrettyPrinter.prototype.append = function(value) {
  this.string += value;
};
jasmine.Queue = function(env) {
  this.env = env;
  this.blocks = [];
  this.running = false;
  this.index = 0;
  this.offset = 0;
  this.abort = false;
};

jasmine.Queue.prototype.addBefore = function(block) {
  this.blocks.unshift(block);
```

```
};


jasmine.Queue.prototype.add = function(block) {
  this.blocks.push(block);
};


jasmine.Queue.prototype.insertNext = function(block) {
  this.blocks.splice((this.index + this.offset + 1), 0, block);
  this.offset++;
};


jasmine.Queue.prototype.start = function(onComplete) {
  this.running = true;
  this.onComplete = onComplete;
  this.next_();
};


jasmine.Queue.prototype.isRunning = function() {
  return this.running;
```

```
};

jasmine.Queue.LOOP_DONT_RECURSE = true;

jasmine.Queue.prototype.next_ = function() {
  var self = this;
  var goAgain = true;

  while (goAgain) {
    goAgain = false;

    if (self.index < self.blocks.length && !this.abort) {
      var calledSynchronously = true;
      var completedSynchronously = false;

      var onComplete = function () {
        if (jasmine.Queue.LOOP_DONT_RECURSE && calledSynchronously) {
          completedSynchronously = true;
          return;
        }

        if (self.blocks[self.index].abort) {
          self.abort = true;
        }

        self.offset = 0;
        self.index++;

        var now = new Date().getTime();
        if (self.env.updateInterval && now - self.env.lastUpdate >
self.env.updateInterval) {
          self.env.lastUpdate = now;
          self.env.setTimeout(function() {
            self.next_();
          }, 0);
        } else {
          if (jasmine.Queue.LOOP_DONT_RECURSE &&
completedSynchronously) {
            goAgain = true;
          } else {
            self.next_();
          }
        }
      };
      self.blocks[self.index].execute(onComplete);

      calledSynchronously = false;
      if (completedSynchronously) {
        onComplete();
      }

    } else {
      self.running = false;
      if (self.onComplete) {
        self.onComplete();
      }
    }
  }
```

```
};


jasmine.Queue.prototype.results = function() {
  var results = new jasmine.NestedResults();
  for (var i = 0; i < this.blocks.length; i++) {
    if (this.blocks[i].results) {
      results.addResult(this.blocks[i].results());
    }
  }
  return results;
};



/**
 * Runner
 *
 * @constructor
 * @param {jasmine.Env} env
 */
```

```
jasmine.Runner = function(env) {

  var self = this;

  self.env = env;

  self.queue = new jasmine.Queue(env);

  self.before_ = [];

  self.after_ = [];

  self.suites_ = [];

};


jasmine.Runner.prototype.execute = function() {

  var self = this;

  if (self.env.reporter.reportRunnerStarting) {

    self.env.reporter.reportRunnerStarting(this);

  }

  self.queue.start(function () {

    self.finishCallback();

  });

};
```

```
jasmine.Runner.prototype.beforeEach = function(beforeEachFunction) {
  beforeEachFunction.typeName = 'beforeEach';
  this.before_.splice(0,0,beforeEachFunction);
};

jasmine.Runner.prototype.afterEach = function(afterEachFunction) {
  afterEachFunction.typeName = 'afterEach';
  this.after_.splice(0,0,afterEachFunction);
};


jasmine.Runner.prototype.finishCallback = function() {
  this.env.reporter.reportRunnerResults(this);
};

jasmine.Runner.prototype.addSuite = function(suite) {
  this.suites_.push(suite);
};
```

```
jasmine.Runner.prototype.add = function(block) {
  if (block instanceof jasmine.Suite) {
    this.addSuite(block);
  }
  this.queue.add(block);
};

jasmine.Runner.prototype.specs = function () {
  var suites = this.suites();
  var specs = [];
  for (var i = 0; i < suites.length; i++) {
    specs = specs.concat(suites[i].specs());
  }
  return specs;
};

jasmine.Runner.prototype.suites = function() {
  return this.suites_;
};
```

```
jasmine.Runner.prototype.topLevelSuites = function() {
  var topLevelSuites = [];
  for (var i = 0; i < this.suites_.length; i++) {
    if (!this.suites_[i].parentSuite) {
      topLevelSuites.push(this.suites_[i]);
    }
  }
  return topLevelSuites;
};


jasmine.Runner.prototype.results = function() {
  return this.queue.results();
};
/**
 * Internal representation of a Jasmine specification, or test.
 *
 * @constructor
 * @param {jasmine.Env} env
 * @param {jasmine.Suite} suite
 * @param {String} description
 */
jasmine.Spec = function(env, suite, description) {
  if (!env) {
    throw new Error('jasmine.Env() required');
  }
  if (!suite) {
    throw new Error('jasmine.Suite() required');
  }
  var spec = this;
  spec.id = env.nextSpecId ? env.nextSpecId() : null;
  spec.env = env;
  spec.suite = suite;
  spec.description = description;
  spec.queue = new jasmine.Queue(env);

  spec.afterCallbacks = [];
  spec.spies_ = [];
  spec.results_ = new jasmine.NestedResults();
  spec.results_.description = description;
  spec.matchersClass = null;
};

jasmine.Spec.prototype.getFullName = function() {
  return this.suite.getFullName() + ' ' + this.description + '.';
};


jasmine.Spec.prototype.results = function() {
  return this.results_;
};

/**
 * All parameters are pretty-printed and concatenated together, then written to the spec's output.
 *
 * Be careful not to leave calls to <code>jasmine.log</code> in production code.
 */
jasmine.Spec.prototype.log = function() {
  return this.results_.log(arguments);
};

jasmine.Spec.prototype.runs = function (func) {
  var block = new jasmine.Block(this.env, func, this);
  this.addToQueue(block);
  return this;
};

jasmine.Spec.prototype.addToQueue = function (block) {
  if (this.queue.isRunning()) {
    this.queue.insertNext(block);
  } else {
    this.queue.add(block);
  }
};
```

```
/**
 * @param {jasmine.ExpectationResult} result
 */
jasmine.Spec.prototype.addMatcherResult = function(result) {
  this.results_.addResult(result);
};

jasmine.Spec.prototype.expect = function(actual) {
  var positive = new (this.getMatchersClass_())(this.env, actual, this);
  positive.not = new (this.getMatchersClass_())(this.env, actual, this, true);
  return positive;
};

/**
 * Waits a fixed time period before moving to the next block.
 *
 * @deprecated Use waitsFor() instead
 * @param {Number} timeout milliseconds to wait
 */
```

```
jasmine.Spec.prototype.waits = function(timeout) {
  var waitsFunc = new jasmine.WaitsBlock(this.env, timeout, this);
  this.addToQueue(waitsFunc);
  return this;
};

/**
 * Waits for the latchFunction to return true before proceeding to the next block.
 *
 * @param {Function} latchFunction
 * @param {String} optional_timeoutMessage
 * @param {Number} optional_timeout
 */
jasmine.Spec.prototype.waitsFor = function(latchFunction, optional_timeoutMessage, optional_timeout) {
  var latchFunction_ = null;
  var optional_timeoutMessage_ = null;
  var optional_timeout_ = null;

  for (var i = 0; i < arguments.length; i++) {
    var arg = arguments[i];
    switch (typeof arg) {
      case 'function':
        latchFunction_ = arg;
        break;
      case 'string':
        optional_timeoutMessage_ = arg;
        break;
      case 'number':
        optional_timeout_ = arg;
        break;
    }
  }

  var waitsForFunc = new jasmine.WaitsForBlock(this.env, optional_timeout_, latchFunction_, optional_timeoutMessage_, this);
  this.addToQueue(waitsForFunc);
  return this;
};
```

```
jasmine.Spec.prototype.fail = function (e) {
  var expectationResult = new jasmine.ExpectationResult({
    passed: false,
    message: e ? jasmine.util.formatException(e) : 'Exception',
    trace: { stack: e.stack }
  });
  this.results_.addResult(expectationResult);
};

jasmine.Spec.prototype.getMatchersClass_ = function() {
  return this.matchersClass || this.env.matchersClass;
};

jasmine.Spec.prototype.addMatchers = function(matchersPrototype) {
  var parent = this.getMatchersClass_();
  var newMatchersClass = function() {
    parent.apply(this, arguments);
  };
  jasmine.util.inherit(newMatchersClass, parent);
  jasmine.Matchers.wrapInto_(matchersPrototype, newMatchersClass);

  this.matchersClass = newMatchersClass;

};


jasmine.Spec.prototype.finishCallback = function() {

  this.env.reporter.reportSpecResults(this);

};


jasmine.Spec.prototype.finish = function(onComplete) {

  this.removeAllSpies();

  this.finishCallback();

  if (onComplete) {

    onComplete();

  }

};


jasmine.Spec.prototype.after = function(doAfter) {

  if (this.queue.isRunning()) {

    this.queue.add(new jasmine.Block(this.env, doAfter, this));
  } else {
    this.afterCallbacks.unshift(doAfter);
  }
};

jasmine.Spec.prototype.execute = function(onComplete) {
  var spec = this;
  if (!spec.env.specFilter(spec)) {
    spec.results_.skipped = true;
    spec.finish(onComplete);
    return;
  }

  this.env.reporter.reportSpecStarting(this);

  spec.env.currentSpec = spec;

  spec.addBeforesAndAftersToQueue();
  spec.queue.start(function () {
    spec.finish(onComplete);
  });
};

jasmine.Spec.prototype.addBeforesAndAftersToQueue = function() {
  var runner = this.env.currentRunner();
  var i;

  for (var suite = this.suite; suite; suite = suite.parentSuite) {
    for (i = 0; i < suite.before_.length; i++) {
      this.queue.addBefore(new jasmine.Block(this.env, suite.before_[i], this));
    }
  }
  for (i = 0; i < runner.before_.length; i++) {
    this.queue.addBefore(new jasmine.Block(this.env, runner.before_[i], this));
  }
  for (i = 0; i < this.afterCallbacks.length; i++) {
    this.queue.add(new jasmine.Block(this.env, this.afterCallbacks[i], this));
  }
  for (suite = this.suite; suite; suite = suite.parentSuite) {
    for (i = 0; i < suite.after_.length; i++) {
      this.queue.add(new jasmine.Block(this.env, suite.after_[i], this));
    }
  }
  for (i = 0; i < runner.after_.length; i++) {
    this.queue.add(new jasmine.Block(this.env, runner.after_[i], this));
  }
};

jasmine.Spec.prototype.explodes = function() {
  throw 'explodes function should not have been called';
};

jasmine.Spec.prototype.spyOn = function(obj, methodName, ignoreMethodDoesntExist) {
  if (obj == jasmine.undefined) {
    throw "spyOn could not find an object to spy upon for " + methodName + "()";
  }
  if (!ignoreMethodDoesntExist && obj[methodName] === jasmine.undefined) {
    throw methodName + '() method does not exist';
  }

  if (!ignoreMethodDoesntExist && obj[methodName] && obj[methodName].isSpy) {
    throw new Error(methodName + ' has already been spied upon');
  }

  var spyObj = jasmine.createSpy(methodName);

  this.spies_.push(spyObj);
  spyObj.baseObj = obj;
  spyObj.methodName = methodName;
  spyObj.originalValue = obj[methodName];

  obj[methodName] = spyObj;

  return spyObj;
```

```
};

jasmine.Spec.prototype.removeAllSpies = function() {
  for (var i = 0; i < this.spies_.length; i++) {
    var spy = this.spies_[i];
    spy.baseObj[spy.methodName] = spy.originalValue;
  }
  this.spies_ = [];
};

/**
 * Internal representation of a Jasmine suite.
 *
 * @constructor
 * @param {jasmine.Env} env
 * @param {String} description
 * @param {Function} specDefinitions
 * @param {jasmine.Suite} parentSuite
 */
```

```
jasmine.Suite = function(env, description, specDefinitions, parentSuite) {
  var self = this;
  self.id = env.nextSuiteId ? env.nextSuiteId() : null;
  self.description = description;
  self.queue = new jasmine.Queue(env);
  self.parentSuite = parentSuite;
  self.env = env;
  self.before_ = [];
  self.after_ = [];
  self.children_ = [];
  self.suites_ = [];
  self.specs_ = [];
};

jasmine.Suite.prototype.getFullName = function() {
  var fullName = this.description;
  for (var parentSuite = this.parentSuite; parentSuite; parentSuite = parentSuite.parentSuite) {
    fullName = parentSuite.description + ' ' + fullName;
  }
  return fullName;
};

jasmine.Suite.prototype.finish = function(onComplete) {
  this.env.reporter.reportSuiteResults(this);
  this.finished = true;
  if (typeof(onComplete) == 'function') {
    onComplete();
  }
};

jasmine.Suite.prototype.beforeEach = function(beforeEachFunction) {
  beforeEachFunction.typeName = 'beforeEach';
  this.before_.unshift(beforeEachFunction);
};

jasmine.Suite.prototype.afterEach = function(afterEachFunction) {
  afterEachFunction.typeName = 'afterEach';
  this.after_.unshift(afterEachFunction);
```

```
};

jasmine.Suite.prototype.results = function() {
  return this.queue.results();
};

jasmine.Suite.prototype.add = function(suiteOrSpec) {
  this.children_.push(suiteOrSpec);
  if (suiteOrSpec instanceof jasmine.Suite) {
    this.suites_.push(suiteOrSpec);
    this.env.currentRunner().addSuite(suiteOrSpec);
  } else {
    this.specs_.push(suiteOrSpec);
  }
  this.queue.add(suiteOrSpec);
};

jasmine.Suite.prototype.specs = function() {
  return this.specs_;
```

```
};


jasmine.Suite.prototype.suites = function() {
  return this.suites_;
};


jasmine.Suite.prototype.children = function() {
  return this.children_;
};


jasmine.Suite.prototype.execute = function(onComplete) {
  var self = this;
  this.queue.start(function () {
    self.finish(onComplete);
  });
};
jasmine.WaitsBlock = function(env, timeout, spec) {
  this.timeout = timeout;
  jasmine.Block.call(this, env, null, spec);
```

```
};


jasmine.util.inherit(jasmine.WaitsBlock, jasmine.Block);


jasmine.WaitsBlock.prototype.execute = function (onComplete) {
  if (jasmine.VERBOSE) {
    this.env.reporter.log('>> Jasmine waiting for ' + this.timeout + ' ms...');
  }
  this.env.setTimeout(function () {
    onComplete();
  }, this.timeout);
};
/**
 * A block which waits for some condition to become true, with timeout.
 *
 * @constructor
 * @extends jasmine.Block
 * @param {jasmine.Env} env The Jasmine environment.
 * @param {Number} timeout The maximum time in milliseconds to wait for the condition to become true.
 * @param {Function} latchFunction A function which returns true when the desired condition has been met.
 * @param {String} message The message to display if the desired condition hasn't been met within the given time period.
 * @param {jasmine.Spec} spec The Jasmine spec.
 */
jasmine.WaitsForBlock = function(env, timeout, latchFunction, message, spec) {
  this.timeout = timeout || env.defaultTimeoutInterval;
  this.latchFunction = latchFunction;
  this.message = message;
  this.totalTimeSpentWaitingForLatch = 0;
  jasmine.Block.call(this, env, null, spec);
};
jasmine.util.inherit(jasmine.WaitsForBlock, jasmine.Block);

jasmine.WaitsForBlock.TIMEOUT_INCREMENT = 10;

jasmine.WaitsForBlock.prototype.execute = function(onComplete) {
  if (jasmine.VERBOSE) {
    this.env.reporter.log('>> Jasmine waiting for ' + (this.message || 'something to happen'));
  }
  var latchFunctionResult;
  try {
    latchFunctionResult = this.latchFunction.apply(this.spec);
  } catch (e) {
    this.spec.fail(e);
    onComplete();
    return;
  }

  if (latchFunctionResult) {
    onComplete();
  } else if (this.totalTimeSpentWaitingForLatch >= this.timeout) {
    var message = 'timed out after ' + this.timeout + ' msec waiting for ' + (this.message || 'something to happen');
    this.spec.fail({
      name: 'timeout',
      message: message
    });

    this.abort = true;
    onComplete();
  } else {
    this.totalTimeSpentWaitingForLatch                                  +=
jasmine.WaitsForBlock.TIMEOUT_INCREMENT;
    var self = this;
    this.env.setTimeout(function() {
      self.execute(onComplete);
    }, jasmine.WaitsForBlock.TIMEOUT_INCREMENT);
  }
};

jasmine.version_= {
  "major": 1,
  "minor": 2,
  "build": 0,
  "revision": 1337005947
};
```

20. assets/www/spec/lib/jasmine-1.2.0/jasmine-html.js

```
jasmine.HtmlReporterHelpers = {};


jasmine.HtmlReporterHelpers.createDom = function(type, attrs, childrenVarArgs)
{
  var el = document.createElement(type);


  for (var i = 2; i < arguments.length; i++) {
    var child = arguments[i];


    if (typeof child === 'string') {
      el.appendChild(document.createTextNode(child));
    } else {
      if (child) {
        el.appendChild(child);
      }
    }
  }
  for (var attr in attrs) {
    if (attr == "className") {
      el[attr] = attrs[attr];
    } else {
      el.setAttribute(attr, attrs[attr]);
    }
  }

  return el;
};

jasmine.HtmlReporterHelpers.getSpecStatus = function(child) {
  var results = child.results();
  var status = results.passed() ? 'passed' : 'failed';
  if (results.skipped) {
    status = 'skipped';
  }

  return status;
```

```
};


jasmine.HtmlReporterHelpers.appendToSummary = function(child, childElement) {
  var parentDiv = this.dom.summary;
  var parentSuite = (typeof child.parentSuite == 'undefined') ? 'suite' : 'parentSuite';
  var parent = child[parentSuite];

  if (parent) {
    if (typeof this.views.suites[parent.id] == 'undefined') {
      this.views.suites[parent.id] = new jasmine.HtmlReporter.SuiteView(parent, this.dom, this.views);
    }
    parentDiv = this.views.suites[parent.id].element;
  }

  parentDiv.appendChild(childElement);
};
```

```
jasmine.HtmlReporterHelpers.addHelpers = function(ctor) {
  for(var fn in jasmine.HtmlReporterHelpers) {
    ctor.prototype[fn] = jasmine.HtmlReporterHelpers[fn];
  }
};

jasmine.HtmlReporter = function(_doc) {
  var self = this;
  var doc = _doc || window.document;

  var reporterView;

  var dom = {};

  // Jasmine Reporter Public Interface
  self.logRunningSpecs = false;

  self.reportRunnerStarting = function(runner) {
```

```
  var specs = runner.specs() || [];

  if (specs.length == 0) {
    return;
  }

  createReporterDom(runner.env.versionString());
  doc.body.appendChild(dom.reporter);

  reporterView = new jasmine.HtmlReporter.ReporterView(dom);
  reporterView.addSpecs(specs, self.specFilter);
};

self.reportRunnerResults = function(runner) {
  reporterView && reporterView.complete();
};

self.reportSuiteResults = function(suite) {
  reporterView.suiteComplete(suite);
  };

  self.reportSpecStarting = function(spec) {
    if (self.logRunningSpecs) {
      self.log('>> Jasmine Running ' + spec.suite.description + ' ' + spec.description
+ '...');
    }
  };

  self.reportSpecResults = function(spec) {
    reporterView.specComplete(spec);
  };

  self.log = function() {
    var console = jasmine.getGlobal().console;
    if (console && console.log) {
      if (console.log.apply) {
        console.log.apply(console, arguments);
      } else {
        console.log(arguments); // ie fix: console.log.apply doesn't exist on ie
    }
  }
};

self.specFilter = function(spec) {
  if (!focusedSpecName()) {
    return true;
  }

  return spec.getFullName().indexOf(focusedSpecName()) === 0;
};

return self;

function focusedSpecName() {
  var specName;

  (function memoizeFocusedSpec() {
    if (specName) {
      return;
    }

    var paramMap = [];
    var params = doc.location.search.substring(1).split('&');

    for (var i = 0; i < params.length; i++) {
      var p = params[i].split('=');
      paramMap[decodeURIComponent(p[0])] = decodeURIComponent(p[1]);
    }

    specName = paramMap.spec;
  })();

  return specName;
}

function createReporterDom(version) {
  dom.reporter = self.createDom('div', { id: 'HTMLReporter', className:
'jasmine_reporter' },
   dom.banner = self.createDom('div', { className: 'banner' },
    self.createDom('span', { className: 'title' }, "Jasmine "),
    self.createDom('span', { className: 'version' }, version)),

   dom.symbolSummary = self.createDom('ul', {className: 'symbolSummary'}),
   dom.alert = self.createDom('div', {className: 'alert'}),
   dom.results = self.createDom('div', {className: 'results'},
    dom.summary = self.createDom('div', { className: 'summary' }),
    dom.details = self.createDom('div', { id: 'details' }))
  );
 }
};
jasmine.HtmlReporterHelpers.addHelpers(jasmine.HtmlReporter);jasmine.HtmlReporter.ReporterView = function(dom) {
 this.startedAt = new Date();
 this.runningSpecCount = 0;
 this.completeSpecCount = 0;
 this.passedCount = 0;
 this.failedCount = 0;
```

```
  this.skippedCount = 0;


  this.createResultsMenu = function() {
    this.resultsMenu = this.createDom('span', {className: 'resultsMenu bar'},
      this.summaryMenuItem = this.createDom('a', {className:
'summaryMenuItem', href: "#"}, '0 specs'),
      ' | ',
      this.detailsMenuItem = this.createDom('a', {className: 'detailsMenuItem',
href: "#"}, '0 failing'));


    this.summaryMenuItem.onclick = function() {
      dom.reporter.className = dom.reporter.className.replace(/ showDetails/g,
'');
    };


    this.detailsMenuItem.onclick = function() {
      showDetails();
    };
  };
  this.addSpecs = function(specs, specFilter) {
    this.totalSpecCount = specs.length;

    this.views = {
      specs: {},
      suites: {}
    };

    for (var i = 0; i < specs.length; i++) {
      var spec = specs[i];
      this.views.specs[spec.id] = new jasmine.HtmlReporter.SpecView(spec, dom,
this.views);
      if (specFilter(spec)) {
        this.runningSpecCount++;
      }
    }
  };

  this.specComplete = function(spec) {
    this.completeSpecCount++;
```

```
if (isUndefined(this.views.specs[spec.id])) {
  this.views.specs[spec.id] = new jasmine.HtmlReporter.SpecView(spec, dom);
}

var specView = this.views.specs[spec.id];

switch (specView.status()) {
  case 'passed':
    this.passedCount++;
    break;

  case 'failed':
    this.failedCount++;
    break;

  case 'skipped':
    this.skippedCount++;
    break;
  }

  specView.refresh();
  this.refresh();
};

this.suiteComplete = function(suite) {
  var suiteView = this.views.suites[suite.id];
  if (isUndefined(suiteView)) {
    return;
  }
  suiteView.refresh();
};

this.refresh = function() {

  if (isUndefined(this.resultsMenu)) {
    this.createResultsMenu();
  }
  // currently running UI
  if (isUndefined(this.runningAlert)) {
    this.runningAlert = this.createDom('a', {href: "?", className: "runningAlert bar"});
    dom.alert.appendChild(this.runningAlert);
  }
  this.runningAlert.innerHTML = "Running " + this.completeSpecCount + " of " + specPluralizedFor(this.totalSpecCount);

  // skipped specs UI
  if (isUndefined(this.skippedAlert)) {
    this.skippedAlert = this.createDom('a', {href: "?", className: "skippedAlert bar"});
  }

  this.skippedAlert.innerHTML = "Skipping " + this.skippedCount + " of " + specPluralizedFor(this.totalSpecCount) + " - run all";

  if (this.skippedCount === 1 && isDefined(dom.alert)) {
    dom.alert.appendChild(this.skippedAlert);
  }

  // passing specs UI
  if (isUndefined(this.passedAlert)) {
    this.passedAlert = this.createDom('span', {href: "?", className: "passingAlert
bar"});
  }
  this.passedAlert.innerHTML = "Passing " +
specPluralizedFor(this.passedCount);

  // failing specs UI
  if (isUndefined(this.failedAlert)) {
    this.failedAlert = this.createDom('span', {href: "?", className: "failingAlert
bar"});
  }
  this.failedAlert.innerHTML = "Failing " + specPluralizedFor(this.failedCount);

  if (this.failedCount === 1 && isDefined(dom.alert)) {
    dom.alert.appendChild(this.failedAlert);
    dom.alert.appendChild(this.resultsMenu);
  }

  // summary info
  this.summaryMenuItem.innerHTML = "" + specPluralizedFor(this.runningSpecCount);
  this.detailsMenuItem.innerHTML = "" + this.failedCount + " failing";
};

this.complete = function() {
  dom.alert.removeChild(this.runningAlert);

  this.skippedAlert.innerHTML = "Ran " + this.runningSpecCount + " of " + specPluralizedFor(this.totalSpecCount) + " - run all";

  if (this.failedCount === 0) {
    dom.alert.appendChild(this.createDom('span', {className: 'passingAlert bar'}, "Passing " + specPluralizedFor(this.passedCount)));
  } else {
    showDetails();
  }
dom.banner.appendChild(this.createDom('span', {className: 'duration'}, "finished in " + ((new Date().getTime() - this.startedAt.getTime()) / 1000) + "s"));
};

return this;

function showDetails() {
if (dom.reporter.className.search(/showDetails/) === -1) {
  dom.reporter.className += " showDetails";
}
}

function isUndefined(obj) {
return typeof obj === 'undefined';
}

function isDefined(obj) {
return !isUndefined(obj);
}
```

```
  function specPluralizedFor(count) {

    var str = count + " spec";

    if (count > 1) {

      str += "s"

    }

    return str;

  }


};


jasmine.HtmlReporterHelpers.addHelpers(jasmine.HtmlReporter.ReporterView);



jasmine.HtmlReporter.SpecView = function(spec, dom, views) {

  this.spec = spec;

  this.dom = dom;

  this.views = views;
  this.symbol = this.createDom('li', { className: 'pending' });
  this.dom.symbolSummary.appendChild(this.symbol);

  this.summary = this.createDom('div', { className: 'specSummary' },
    this.createDom('a', {
      className: 'description',
      href: '?spec=' + encodeURIComponent(this.spec.getFullName()),
      title: this.spec.getFullName()
    }, this.spec.description)
  );

  this.detail = this.createDom('div', { className: 'specDetail' },
    this.createDom('a', {
      className: 'description',
      href: '?spec=' + encodeURIComponent(this.spec.getFullName()),
      title: this.spec.getFullName()
    }, this.spec.getFullName())
  );
};
```

```
jasmine.HtmlReporter.SpecView.prototype.status = function() {
  return this.getSpecStatus(this.spec);
};

jasmine.HtmlReporter.SpecView.prototype.refresh = function() {
  this.symbol.className = this.status();

  switch (this.status()) {
    case 'skipped':
      break;

    case 'passed':
      this.appendSummaryToSuiteDiv();
      break;

    case 'failed':
      this.appendSummaryToSuiteDiv();
      this.appendFailureDetail();
      break;
  }
};

jasmine.HtmlReporter.SpecView.prototype.appendSummaryToSuiteDiv = function() {
  this.summary.className += ' ' + this.status();
  this.appendToSummary(this.spec, this.summary);
};

jasmine.HtmlReporter.SpecView.prototype.appendFailureDetail = function() {
  this.detail.className += ' ' + this.status();

  var resultItems = this.spec.results().getItems();
  var messagesDiv = this.createDom('div', { className: 'messages' });

  for (var i = 0; i < resultItems.length; i++) {
    var result = resultItems[i];

    if (result.type == 'log') {
      messagesDiv.appendChild(this.createDom('div', {className: 'resultMessage log'}, result.toString()));
    } else if (result.type == 'expect' && result.passed && !result.passed()) {
      messagesDiv.appendChild(this.createDom('div', {className: 'resultMessage fail'}, result.message));

      if (result.trace.stack) {
        messagesDiv.appendChild(this.createDom('div', {className: 'stackTrace'}, result.trace.stack));
      }
    }
  }

  if (messagesDiv.childNodes.length > 0) {
    this.detail.appendChild(messagesDiv);
    this.dom.details.appendChild(this.detail);
  }
};

jasmine.HtmlReporterHelpers.addHelpers(jasmine.HtmlReporter.SpecView);jasmine.HtmlReporter.SuiteView = function(suite, dom, views) {
  this.suite = suite;
  this.dom = dom;
  this.views = views;

  this.element = this.createDom('div', { className: 'suite' },
    this.createDom('a', { className: 'description', href: '?spec=' + encodeURIComponent(this.suite.getFullName()) }, this.suite.description)
  );

  this.appendToSummary(this.suite, this.element);
};

jasmine.HtmlReporter.SuiteView.prototype.status = function() {
  return this.getSpecStatus(this.suite);
};

jasmine.HtmlReporter.SuiteView.prototype.refresh = function() {
  this.element.className += " " + this.status();
};
```

```
jasmine.HtmlReporterHelpers.addHelpers(jasmine.HtmlReporter.SuiteView);


/* @deprecated Use jasmine.HtmlReporter instead
 */
jasmine.TrivialReporter = function(doc) {
  this.document = doc || document;
  this.suiteDivs = {};
  this.logRunningSpecs = false;
};


jasmine.TrivialReporter.prototype.createDom = function(type, attrs, childrenVarArgs) {
  var el = document.createElement(type);


  for (var i = 2; i < arguments.length; i++) {
    var child = arguments[i];


    if (typeof child === 'string') {
      el.appendChild(document.createTextNode(child));
    } else {
```

```
    if (child) { el.appendChild(child); }
   }
  }

  for (var attr in attrs) {
   if (attr == "className") {
    el[attr] = attrs[attr];
   } else {
    el.setAttribute(attr, attrs[attr]);
   }
  }

  return el;
};

jasmine.TrivialReporter.prototype.reportRunnerStarting = function(runner) {
  var showPassed, showSkipped;

  this.outerDiv = this.createDom('div', { id: 'TrivialReporter', className:
'jasmine_reporter' },
    this.createDom('div', { className: 'banner' },
     this.createDom('div', { className: 'logo' },
        this.createDom('span', { className: 'title' }, "Jasmine"),
        this.createDom('span', { className: 'version' },
runner.env.versionString())),
     this.createDom('div', { className: 'options' },
        "Show ",
        showPassed = this.createDom('input', { id:
"__jasmine_TrivialReporter_showPassed__", type: 'checkbox' }),
        this.createDom('label', { "for":
"__jasmine_TrivialReporter_showPassed__" }, " passed "),
        showSkipped = this.createDom('input', { id:
"__jasmine_TrivialReporter_showSkipped__", type: 'checkbox' }),
        this.createDom('label', { "for":
"__jasmine_TrivialReporter_showSkipped__" }, " skipped")
        )
      ),

    this.runnerDiv = this.createDom('div', { className: 'runner running' },
      this.createDom('a', { className: 'run_spec', href: '?' }, "run all"),
      this.runnerMessageSpan = this.createDom('span', {}, "Running..."),
          this.finishedAtSpan = this.createDom('span', { className: 'finished-at' },
""))
      );

  this.document.body.appendChild(this.outerDiv);

  var suites = runner.suites();
  for (var i = 0; i < suites.length; i++) {
    var suite = suites[i];
    var suiteDiv = this.createDom('div', { className: 'suite' },
        this.createDom('a', { className: 'run_spec', href: '?spec=' +
encodeURIComponent(suite.getFullName()) }, "run"),
        this.createDom('a', { className: 'description', href: '?spec=' +
encodeURIComponent(suite.getFullName()) }, suite.description));
    this.suiteDivs[suite.id] = suiteDiv;
    var parentDiv = this.outerDiv;
    if (suite.parentSuite) {
      parentDiv = this.suiteDivs[suite.parentSuite.id];
    }
    parentDiv.appendChild(suiteDiv);
  }

  this.startedAt = new Date();

  var self = this;
  showPassed.onclick = function(evt) {
    if (showPassed.checked) {
      self.outerDiv.className += ' show-passed';
    } else {
      self.outerDiv.className = self.outerDiv.className.replace(/ show-passed/, '');
    }
  };

  showSkipped.onclick = function(evt) {
    if (showSkipped.checked) {
      self.outerDiv.className += ' show-skipped';
    } else {
      self.outerDiv.className = self.outerDiv.className.replace(/ show-skipped/, '');
    }
  };
};

jasmine.TrivialReporter.prototype.reportRunnerResults = function(runner) {
  var results = runner.results();
  var className = (results.failedCount > 0) ? "runner failed" : "runner passed";
  this.runnerDiv.setAttribute("class", className);
  //do it twice for IE
  this.runnerDiv.setAttribute("className", className);
  var specs = runner.specs();
  var specCount = 0;
  for (var i = 0; i < specs.length; i++) {
    if (this.specFilter(specs[i])) {
      specCount++;
    }
  }
  var message = "" + specCount + " spec" + (specCount == 1 ? "" : "s" ) + ", " + results.failedCount + " failure" + ((results.failedCount == 1) ? "" : "s");
  message += " in " + ((new Date().getTime() - this.startedAt.getTime()) / 1000) +
"s";

  this.runnerMessageSpan.replaceChild(this.createDom('a', { className:
'description', href: '?'}, message), this.runnerMessageSpan.firstChild);


  this.finishedAtSpan.appendChild(document.createTextNode("Finished at " +
new Date().toString()));

};


jasmine.TrivialReporter.prototype.reportSuiteResults = function(suite) {

  var results = suite.results();

  var status = results.passed() ? 'passed' : 'failed';

  if (results.totalCount === 0) { // todo: change this to check results.skipped

    status = 'skipped';

  }

  this.suiteDivs[suite.id].className += " " + status;

};


jasmine.TrivialReporter.prototype.reportSpecStarting = function(spec) {

  if (this.logRunningSpecs) {
    this.log('>> Jasmine Running ' + spec.suite.description + ' ' + spec.description + '...');
  }
};

jasmine.TrivialReporter.prototype.reportSpecResults = function(spec) {
  var results = spec.results();
  var status = results.passed() ? 'passed' : 'failed';
  if (results.skipped) {
    status = 'skipped';
  }
  var specDiv = this.createDom('div', { className: 'spec '  + status },
      this.createDom('a', { className: 'run_spec', href: '?spec=' + encodeURIComponent(spec.getFullName()) }, "run"),
      this.createDom('a', {
        className: 'description',
        href: '?spec=' + encodeURIComponent(spec.getFullName()),
        title: spec.getFullName()
      }, spec.description));
  var resultItems = results.getItems();
  var messagesDiv = this.createDom('div', { className: 'messages' });
  for (var i = 0; i < resultItems.length; i++) {
    var result = resultItems[i];

    if (result.type == 'log') {
      messagesDiv.appendChild(this.createDom('div', {className: 'resultMessage log'}, result.toString()));
    } else if (result.type == 'expect' && result.passed && !result.passed()) {
      messagesDiv.appendChild(this.createDom('div', {className: 'resultMessage fail'}, result.message));

      if (result.trace.stack) {
        messagesDiv.appendChild(this.createDom('div', {className: 'stackTrace'}, result.trace.stack));
      }
    }
  }
  if (messagesDiv.childNodes.length > 0) {
    specDiv.appendChild(messagesDiv);
  }

  this.suiteDivs[spec.suite.id].appendChild(specDiv);
};

jasmine.TrivialReporter.prototype.log = function() {
  var console = jasmine.getGlobal().console;
  if (console && console.log) {
    if (console.log.apply) {
      console.log.apply(console, arguments);
    } else {
      console.log(arguments); // ie fix: console.log.apply doesn't exist on ie
    }
  }
};

jasmine.TrivialReporter.prototype.getLocation = function() {
  return this.document.location;
};

jasmine.TrivialReporter.prototype.specFilter = function(spec) {
  var paramMap = {};
  var params = this.getLocation().search.substring(1).split('&');
  for (var i = 0; i < params.length; i++) {
    var p = params[i].split('=');
    paramMap[decodeURIComponent(p[0])] = decodeURIComponent(p[1]);
  }

  if (!paramMap.spec) {
    return true;
  }
  return spec.getFullName().indexOf(paramMap.spec) === 0;
};
```