Widget Packaging and XML Configuration use in PhoneGap Hybrid Mobile Applications

Filip Maj
Michael Brooks

Nitobi Software Inc.
Vancouver, B.C., Canada

Hybrid applications, or mobile applications that employ a combination of native platform as well as web-based source code, are becoming a more viable development avenue in an increasing number of cases. Facebook and The New York Times are two prominent examples that are available in a few different mobile application stores and are built in this manner. PhoneGap (a.k.a. Apache Callback as an Apache Incubator project) is a popular cross-platform development framework that enables developers to write their applications almost solely in web-standards based code. The framework has different hybrid implementations for seven of the most popular mobile platforms: iOS, Android, BlackBerry WebWorks, Windows Phone 7, Samsung Bada, webOS and Symbian.

The W3C Widget specification, and specifically the XML Configuration portion of it, is extremely relevant to the PhoneGap project. The project is still quite young, but fully utilizing a config.xml file to specify application metadata and configuration parameters is beginning to be incorporated. The core PhoneGap project closely follows section 6 (Widget Packages) of the XML Configuration specification as a guide for the directory structure of a PhoneGap application's web assets. Additionally, PhoneGap Build, Nitobi's cloud-based PhoneGap-building service that takes hybrid application web assets and compiles them into different native application binaries, has already been using the config.xml file extensively to handle the brunt of application data and configuration tasks for the end user. This paper aims to show areas where our team had to extend the specification, highlight some of the main incompatibilities and inconsistencies and offer suggestions on how to improve the W3C Widget Packaging and XML Configuration specification moving forward.

Use in PhoneGap Build

PhoneGap Build boils down to a service where users can upload PhoneGap web assets (the HTML, CSS and JavaScript making up the majority of their application) which get wrapped up in the various PhoneGap implementations for each platform, compiled, and sent back to the user.

The service allows for the inclusion of a config.xml file in the root of packages being uploaded (see https://build.phonegap.com/docs/config-xml for details). Behind the scenes, PhoneGap Build will use the provided config.xml file and convert it to different files for each platform to preserve the specified configuration parameters. For details on the process you can check out Andrew Lunny's open-source confetti project, which deals with this single problem. The existing specification does a very good job of covering the majority of use cases, however, there are some issues.

- <widget id> attribute: Currently on build.phonegap.com we recommend employing a reverse domain style ID, i.e. com.yourdomain.yourapp. The id will get converted into whatever format is required for the various target platforms. Most will accept the reverse domain style, but some (for example Windows Phone 7) require a different format (a GUID).
- <widget version> attribute: our use of this attribute specifies the version *string* of the application, which is likely not the *intended* use of this attribute. We recommend employing a Java-style version (major/minor/patch) of the form 1.0.0, as some platforms require this format (webOS).
  Related, PhoneGap Build employs an additional versionCode attribute (specifically for Android) that would be used as a monotonically-increasing number between application revisions. This is used by the Android Market to inform the user when new versions of Android apps are published. Build will simply use the total build count for an application for this attribute if none is specified.
- <feature> element is an excellent first step and maps very elegantly to the resolution of APIs that the application will have access to (at run or build time). In PhoneGap's case, these are device APIs such as accelerometer, contacts, or file system access (which are also inspired and guided by the DAP). However, the spec is somewhat incomplete. Taken from the spec:

  "How a user agent makes use of features depends on the user agent's security policy, hence activation and authorization requirements for features are beyond the scope of this specification."

  Perhaps this is something worth discussing, as Build employs a combination of <feature> name URI mapping to the applications-specific permissions that must be approved on a per-application basis for certain platforms (BlackBerry, Android) by the user. On the flip side, some platforms such as iOS require permission resolution at run time. The two approaches are at odds with each other. Can the config.xml file specification, as an abstraction, be structured differently to facilitate its use as such?
- <preference> element. Currently in use for three purposes on Build:
  - orientation specification. You can set an app to be locked to a specific orientation.
  - target-device. Can specify handset, tablet or universal.
  - permissions. Can be specified with a value of "none" and this will translate over to no required permissions for Android applications. This is extremely unwieldy

as we already do something similar to this through the <feature> element declarations.

One new element that we added to the config.xml file is <gap:splash>. It defines a splash screen for the app. Under the hood, there is a parent construct (“image”) which has child constructs for <icon> and <gap:splash> elements. One idea we had was to use a “type” and/or “encoding” attribute with a general “image” element to support defining both icons and other types of images used throughout the application. On the flip side, there is a <content> element available in the spec that also allows for specifying these various media types. Is there an opportunity to consolidate these? If not, can we expand on the specification to provide a better definition for the <content> and <icon> elements to differentiate between the two more easily?

One thing worth mentioning would be how RIM, makers of the BlackBerry, has embraced the config.xml file and the W3C Widget specification in their WebWorks SDK. WebWorks is a web-based abstraction that RIM provides on top of their BlackBerry Java SDK to enable web developers to build applications for the BlackBerry using nothing but web-based code. In short: it is the same as PhoneGap. RIM uses all of the usual config.xml elements, but also provides a few extensions of their own, wrapped up in their own XML namespace, to allow the developer to define application parameters specific to the BlackBerry platform. A few examples of some of the extensions they employ:

- a rim:hover attribute on <icon> elements, for specifying the hover state on the application button within the system-wide apps menu.
- a rim:navigation attribute for specifying navigation style.
- a <rim:cache> element to specify application caching behaviours.
- a <rim:connection> element to specify what kind of network to use for data connectivity.
- a rim:backButton attribute for specifying hardware back button behaviour.

Problems That Still Need Solving

One thing that could be useful is better support for targeting specific platforms, with specific features and preferences. We are not sure of a suitable structure - either having platforms as child elements of features/preferences, or having a top-level platform element, with feature/preference elements as children, seem reasonable. Some kind of hierarchy between features and platforms makes sense as this would elegantly solve the issue of what features are available on what platforms: the developer simply specifies the relevant features for any target platform (or vice versa).