

*Developing Applications in JavaScript  
using the Palm Mojo™ Framework*

# Palm webOS™

**Rough Cuts**

**O'REILLY®**

*Mitch Allen*

## **Palm webOS**

by Mitch Allen

Copyright © 2009 Palm, Inc.. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://safari.oreilly.com>). For more information, contact our corporate/institutional sales department: (800) 998-9938 or [corporate@oreilly.com](mailto:corporate@oreilly.com).

**Editor:** Steve Weiss

**Production Editor:** Molly Sharp

**Copyeditor:**

**Proofreader:**

**Indexer:**

**Cover Designer:**

**Interior Designer:**

**Illustrators:** Robert Romano and Jessamyn Read

### **Printing History:**

August 2009: First Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *Palm webOS* and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-0-596-15525-4  
1234383386

This excerpt is protected by copyright law. It is your responsibility to obtain permissions necessary for any proposed use of this material. Please direct your inquiries to [permissions@oreilly.com](mailto:permissions@oreilly.com).

---

# Table of Contents

<b>1. Overview of webOS .....</b>	<b>1</b>
Application Model	1
Application Framework and OS	2
User Interface	3
Navigation	3
Launcher	4
Card View	5
Notifications and the Dashboard	6
User Interface Principles	9
Mojo Application Framework	11
Anatomy of a webOS Application	11
UI Widgets	15
Services	17
Palm webOS Architecture	17
Application Environment	18
Core OS	19
Software Developer Kit (SDK)	20
Development Tools	20
Mojo Framework and Sample Code	21
Developer Portal	21
Summary	21

---

# Overview of webOS

Palm® webOS™ is Palm's next generation operating system. Designed around an incredibly fast and beautiful user experience and optimized for the multi-tasking user, webOS integrates the power of a window-based operating system with the simplicity of a browser. Applications are built using standard web technologies and languages, but have access to device-based services and data.

Palm webOS is designed to run on a variety of hardware with different screen sizes, resolutions and orientations, with or without keyboards and works best with a touch-panel though doesn't require one. Because the user interface and application model are built around a web browser, the range of suitable hardware platforms is quite wide, requiring only a CPU, some memory, a wireless data connection, a display, and a means for interacting with the UI and entering text.

You can think of webOS applications as native applications, but built from the same standard HTML, CSS and JavaScript that you'd use to develop web applications. Palm has extended the standard web development environment through a JavaScript framework that gives standardized UI widgets, and access to selected device hardware and services.

The user experience is optimized for launching and managing multiple applications at once. WebOS is designed around multi-tasking, and makes it utterly simple to run background applications, to switch between applications in a single step, and to easily handle interruptions and events without losing context.

You will build WebOS applications with common web development tools following typical design and implementation practices for Ajax applications. But your webOS applications are installed and run directly on the device, just as you are used to doing with native applications.

## Application Model

As shown in Figure 1, the original Palm OS has a typical native application model, as do many of the popular mobile operating systems. Under this model the application's data, logic and user interface are integrated within an executable installed on the native operating system, with direct access to the operating system's services and data.

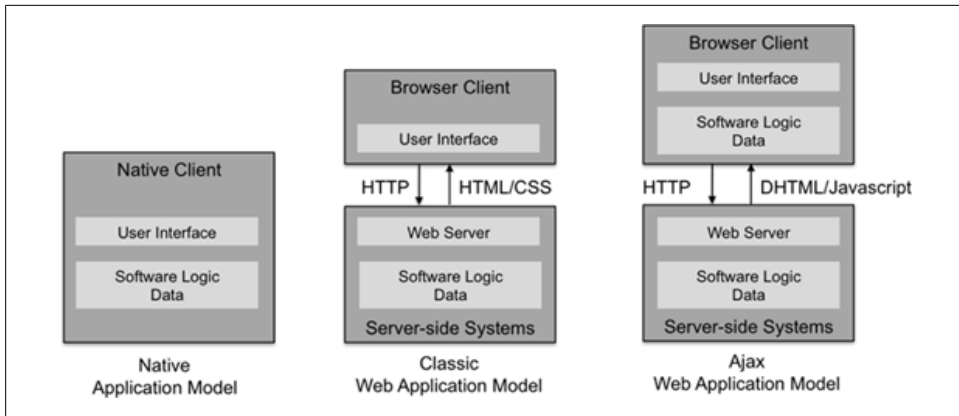


Figure 1. Native and Web Application Models

Classic web applications are basic HTML-based applications that submit an HTTP request to a web server after every user action, and wait for a response before displaying an updated HTML page. More common in recent years are Ajax applications, which handle many user interactions directly and make web server requests asynchronously. As a result, Ajax applications are able to deliver a richer and more responsive user experience. Some of the best examples of this richer experience are the map applications, which enable users to pan and zoom while asynchronously retrieving needed tiles from the web server.

Ajax applications have some significant advantages over embedded applications. They can be more easily deployed and updated through the same search and access techniques used for web pages. Developing web applications is far easier too; the simplicity of the languages and tools, particularly for building *connected* applications, allows developers and designers to be more productive. Connected applications, or applications that leverage dynamic data or web services, are becoming the predominant form for modern applications.

The webOS application model combines the ease of development and maintenance of a web application with the deep integration available to native applications, significantly advancing the mobile user experience, while keeping application development simple.

## Application Framework and OS

Through Palm's application framework, applications can embed UI widgets with sophisticated editing, navigation and display features, enabling more sophisticated application user interfaces. The framework also includes event handling, notification services and a multi-tasking model. Applications can run in the background, managing data, events and services behind the scenes while engaging the user when needed.

You can create and manage your own persistent data using HTML5 storage functions, and you can access data from some of webOS's core applications, such as Contacts and Calendar. You also have access to some basic system services, most of which are device-resident, such as Location services and Accelerometer data, along with some cloud services, such as XMPP messaging.

Architecturally, Palm webOS is an embedded Linux operating system that hosts a custom User Interface (UI) System Manager built on standard browser technology. The System Manager provides a full range of system user interface features including: navigation, application launching and lifecycle management, event management and notifications, system status, local and web searches, and rendering application HTML/CSS/JavaScript code.

You don't need to build a webOS application to make your web content accessible to webOS devices. Palm webOS has a separate browser application to handle standard web pages, and browser-based web applications. While it's expected that more and more web content and services will be delivered as webOS applications, there are millions of legacy websites and information that will continue to be presented in ways best viewed with a classic web browser. Palm webOS supports traditional web content very competitively.

Beyond the operating system, webOS includes a number of core applications: contacts, calendar, tasks, memos, phone, browser, email and messaging. Other applications are included in the initial release, such as a camera, photo viewer, audio/video player and map application, but the full application suite for a given webOS device will vary depending on the model and carrier configuration.

## User Interface

Palm webOS is designed for mobile, battery-operated devices with limited though variable screen sizes, and a touch-driven user interface. User interaction is centered on one application at a time, though applications, once launched, continue to run until closed even when moved out of the foreground view. There is a rich notification system enabling applications to subtly inform or directly engage the user, at the application's discretion.

## Navigation

Navigation is based upon a few simple *gestures* with optional extensions that create a rich vocabulary of commands to drive powerful navigation and editing features. To start with though, all you need to know is:

- *tap* (act on the indicated object). Commonly in a view that contains clusters or lists of items, tapping reveals information contained in an item. This can be thought of as an *open* function, which changes the nature or context of the view to be about



Figure 2. Quick Launch bar and Launcher

the selected item exclusively. Alternately, a tap will change an object's state such as setting a checkbox or selecting an object.

- *back* (the inverse of *open*). This feature looks like the opposite of a tap: the item compresses down to its summary in the containing context where it belongs. Typically, it reverses a view transition, as going from a child view to a parent view.
- *scroll* (flick and quick drags are used to scroll through lists and other views).

Beyond this, you can learn to pan, zoom, drag & drop, switch applications, switch views, search, filter lists and launch applications. But to begin with, only these three gestures are needed to use a webOS device.

## Launcher

When you turn on a webOS device, the screen displays the selected wallpaper image with the *status bar* across the top of the screen and, hovering near the bottom, the *Quick Launch bar*. The Quick Launch bar is used to start up favorite applications or to bring up the *Launcher* for access to all applications on the device. From this view, a search can be initiated simply by typing the search string; searches can be performed on contacts, installed applications, or to start a web search. Figure 2 shows both the Quick Launch bar and the Launcher.

The launched application takes over the available screen becoming the *foreground* application; the application's view replaces the wallpaper image and the Quick Launch

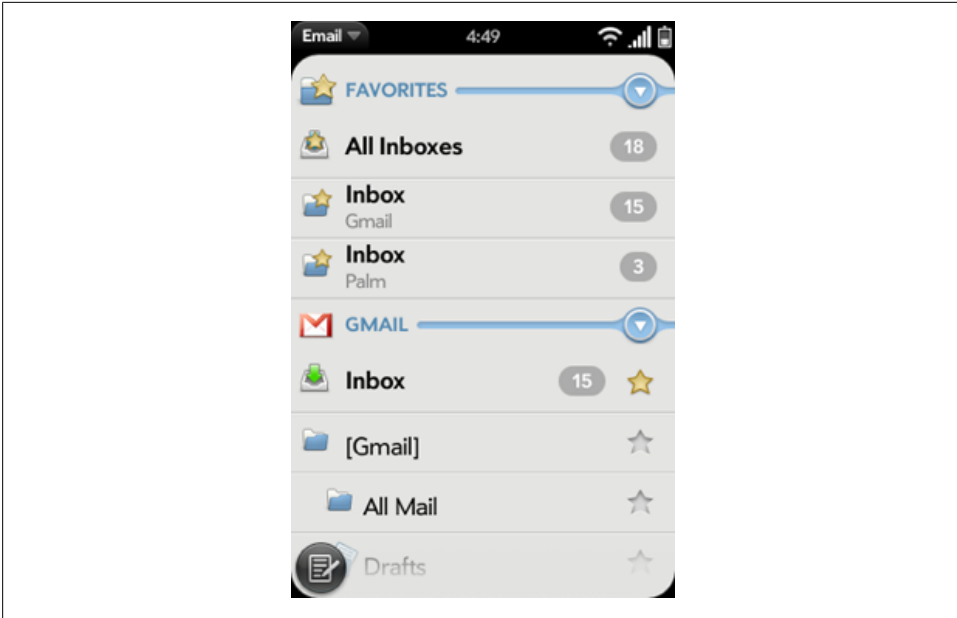


Figure 3. Email Application

bar is dismissed. The status bar remains and is always visible except for full screen mode, which is available to applications such as the video player, or others that request it. This sequence is fluid and smooth, as you will see with all webOS transitions.

## Card View

Figure 3 shows an application's main view, in this case the email application's folder view. The application view includes UI elements that make up the basic email application, in this case the inbox view displays specific folders, which when selected will open a new card with a detail view of the messages contained within the selected folder. At the bottom, the floating icons that you see are menu items. A tap to the menu icons will typically reveal another view associated with the menu action, a sub-menu or a dialog.

But running one application at a time, or performing one *activity* at a time, can be terribly restrictive, and inefficient. Palm webOS was designed to make it easy to work on more than one thing at a time. Simply pressing the *Center button* brings up a new view, the *Card view*, an example of which is shown in Figure 4. From the Card view, you can switch to another activity simply by scrolling to and tapping the card representing the activity. Alternately, you can launch another application from the Quick Launch bar.

The Card view was inspired by the way one handles a deck of cards. Cards can be fanned out to see what card is where. Within a deck of cards, any single card can be

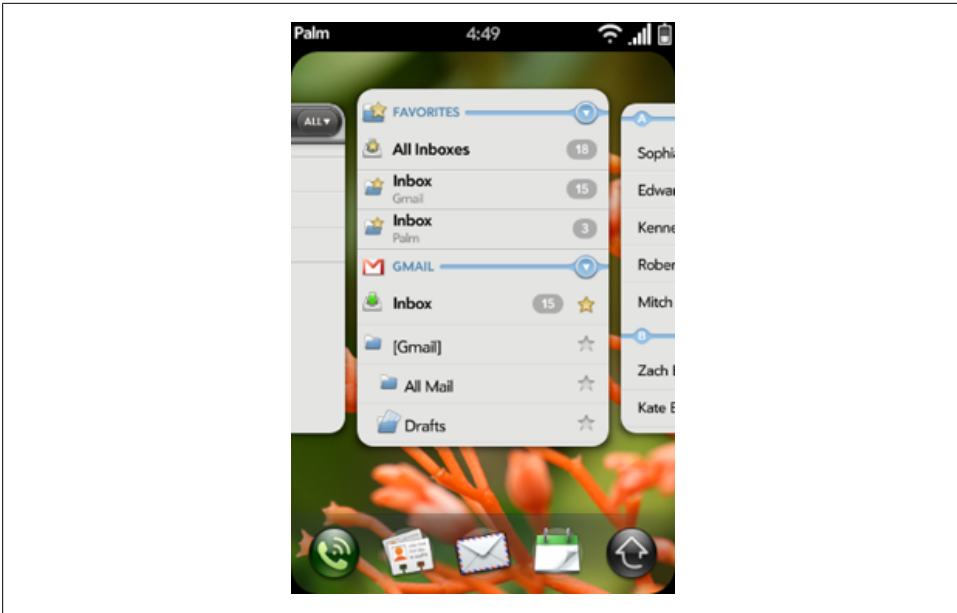


Figure 4. Card view with Email app and other apps

selected or removed with a simple gesture, or moved to a new location by slipping it between adjacent cards. The webOS Card view can be manipulated in similar ways by scrolling through the cards, selecting and flicking cards off the top to remove them or selecting and dragging a card to a new location.

We've introduced the term *activity*, which needs further explanation. Often by design, you will work on one activity at a time with many applications, but with some applications it is more natural to work on several activities in parallel. A common email activity is writing a new email, but in the middle of writing that email, you may want to return to the inbox to look up some information in another email or perhaps read an urgent mail that has just arrived.

With a webOS device, the draft email has its own card separate from the email inbox card. In fact, you can have as many draft emails, each in their own card, as you need; each is considered a separate activity and independently accessible. Switching between emails is as simple as switching between applications and your data is safe, as it is always saved. Figure 5 shows the Card view with the Email application's inbox card and a draft email compose card.

## Notifications and the Dashboard

What happens to the current foreground application when you switch to a new application? The previous application is not closed but continues to run as a *background* application. Background applications can get events, read and write data, access serv-

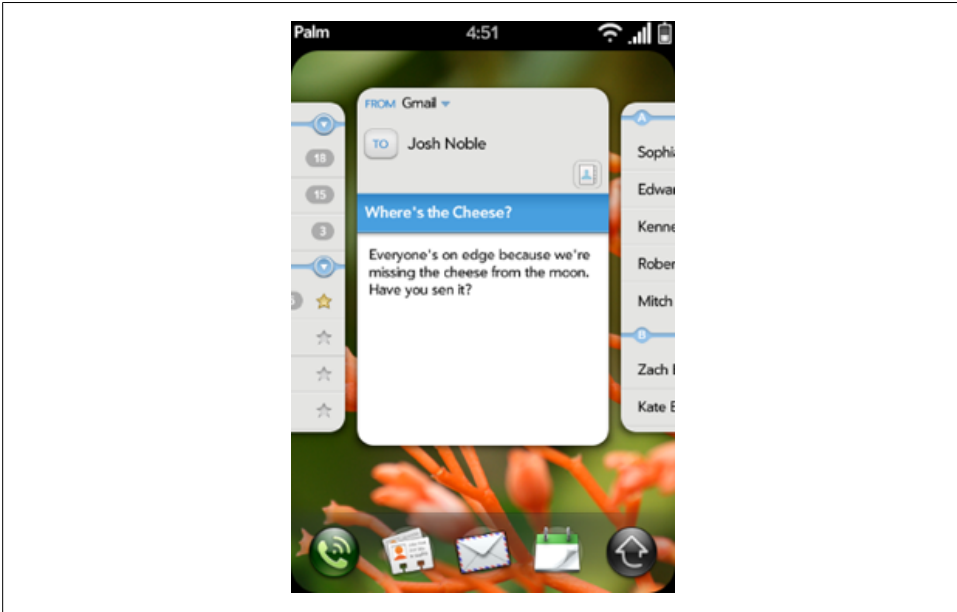


Figure 5. Card view with Email Application and New Email

ices, repaint themselves and are generally not restricted other than to run at a lower priority than the foreground application.

To enable background applications to communicate with the user, Palm provides a notification system with two types of notifications:

- *Popup*. Non-modal dialogs which are of fixed height and include at least one button to dismiss the dialog
- *Banner*. Non-modal icon and single non-styled string of text

Popup notifications are disruptive, appropriate for incoming phone calls, calendar alarms, navigation notifications and other time sensitive or urgent notifications. Users are forced to take action with pop-ups or explicitly clear them but since they are not modal, users are not required to respond immediately.

Banner notifications are displayed in a slow crawl along the bottom of the screen within the *Notification bar*, which sits just below the application window in what is called *negative space* since it is outside of the card's window. After being displayed, banner notifications can selectively leave a *summary icon* in the Notification bar as a reminder to the user. Figure 6 shows an example of a banner notification and the summary icons are shown in Figure 7 indicating that the music player is active and that there is an upcoming calendar event and new messages.

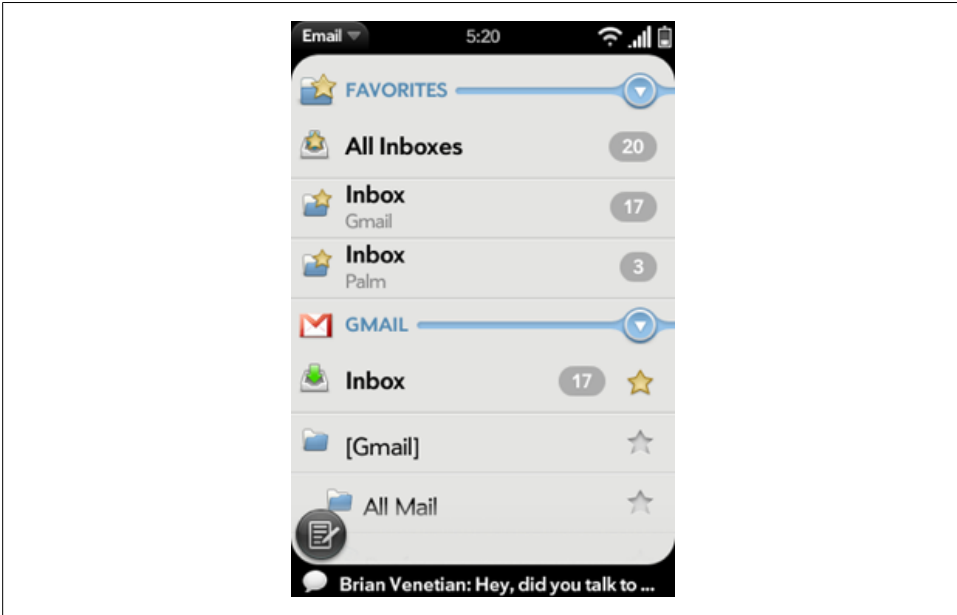


Figure 6. Banner Notification

At any time, the user can tap the Notification bar, which brings up the *dashboard*, shown in Figure 8. Notifications that are not cleared will display their current status within a dashboard *panel*.

Dashboard panels though are more than just summaries of notifications but are dynamic views enabling any background application to display ambient information or status. For example, the calendar application always displays the next event on the calendar even before the event notification has been issued. In Figure 8, the Music application shows the current song along with playback controls that you can manipulate to pause the music or change the selection.

The Notification Bar and Dashboard manage interruptions and events, keeping you abreast of changes in information without disrupting your current activities. It may help to think of them as an event-driven model for viewing and managing your world, while the Card view provides you with task-oriented navigation tools. The combination gives you a few powerful tools from which you can quickly track and access what you need when you need it.

*Headless* applications are those that can be completely served through the dashboard, as their entire purpose is to monitor and present information. For example, a weather application could display the current weather for a targeted location in a dashboard without having a card view at all.

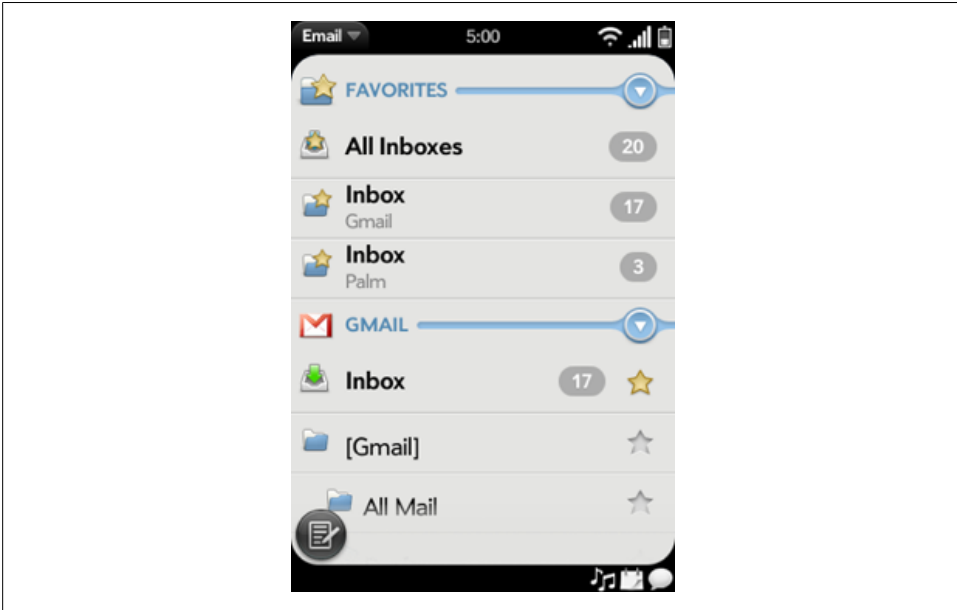


Figure 7. Notification Icons

You will tend to use the Card view to switch between tasks, launch applications and otherwise perform activities. Dashboard is used to monitor your world, to see what's changed or what events have taken place, which will often drive new activities.

## User Interface Principles

There are some foundational principles or values that support the overall webOS user experience; application designers can exploit these same principles to more deeply integrate the application into the overall device experience and enhance the user's experience. Developers can rely on the framework to provide most of what is required at an implementation level, but the application design should anticipate these needs.

Here are the key principles to keep in mind while designing your application:

- *Physical metaphors* are reinforced through direct interaction with application objects and features, instant response to actions, followed by smooth display and object transitions with physics-based scrolling and other movement. For example, objects are deleted by flicking off screen and editing is in place without auxiliary dialogs or scenes.
- Maintain a *sense of place* with repeatable actions, reversible actions, stable object placement and visual transitions taking the user from one place to the next.
- Always display *up-to-date data*, which requires both pushing and pulling the latest data onto the device so that the user is never looking at stale data when more recent

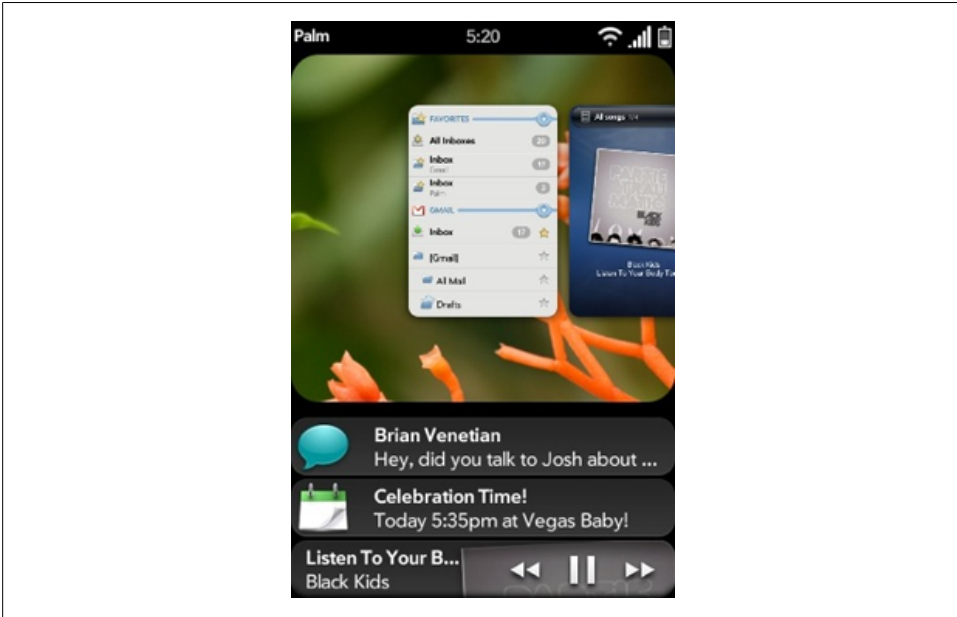


Figure 8. Dashboard

data is available. But this also means managing on-device caches so that when the device is out of coverage or otherwise off-line, the user has access to the last data received.

- Palm webOS is *fast and simple* to use; all features should be designed for instant response, easy for novices to learn while efficient for experienced users.
- *Minimize the steps* for all common functions; put frequently executed commands on the screen, the next most frequent under the menus. Avoid preferences and settings where possible and where not, keep them minimal.
- *Don't block the user*; don't use a modal control when the same function can be carried out non-modally.
- *Be consistent*; help the user learn new tasks and features by leveraging what they have already learned.

Palm applications have always been built around a direct interaction model, where the user touches the screen to select, navigate, and edit. Palm webOS applications have a significantly expanded vocabulary for interaction, but they start at the same place. Your application design should be strongly centered on direct interaction, with clear and distinguishable targets. The platform will provide physical metaphors through display and navigation, but applications need to extend the metaphor with instantaneous response to user actions, to smoothly transitioning display changes, and object transitions.

You can find a lot more on the user interface guidelines and design information in the Palm webOS SDK under the Design Guide. We'll touch on the principles and reference standard style guidelines in the next few chapters, but will not be covering this topic in depth.

## Mojo Application Framework

A webOS application is similar to a web application based on standard HTML, CSS, and JavaScript, but the application lifecycle is different. Applications are run within the UI System Manager, an application runtime built on standard browser technology, to render the display, assist with events, and handle JavaScript.

The webOS APIs are delivered as a JavaScript framework, called Mojo, which supports common application-level functions, UI widgets, access to built-in applications and their data, and native services. To build full-featured webOS applications, many developers will also leverage HTML5 features such as video/audio tagging and database functions. Although not formally part of the framework, the **Prototype** JavaScript framework is bundled with Mojo to assist with registering for events and DOM handling among many other great features.

The framework provides a specific structure for applications to follow based on the Model-View-Controller (MVC) architectural pattern. This allows for better separation of business logic, data, and presentation. Following the conventions reduces complexity; each component of an application has a defined format and location that the framework knows how to handle by default.

You will get a more extensive overview of Mojo in Chapter 2, and details on widgets, services and styles starting in Chapter 3. For now, you should know that the framework includes:

- *Application structure*, such as controllers, views, models, events, storage, notifications, logging and asserts;
- *UI widgets*, including simple single-function widgets, complex multi-function widgets and integrated media viewers;
- *Services*, including access to application data and cross-app launching, storage services, location services, cloud services, and accelerometer data;

## Anatomy of a webOS Application

Outside of the built-in applications, webOS applications are deployed over the web. They can be found in Palm's *App Catalog*, an application distribution service, built into all webOS devices and available to all registered developers. The basic lifecycle stages are illustrated in Figure 9.

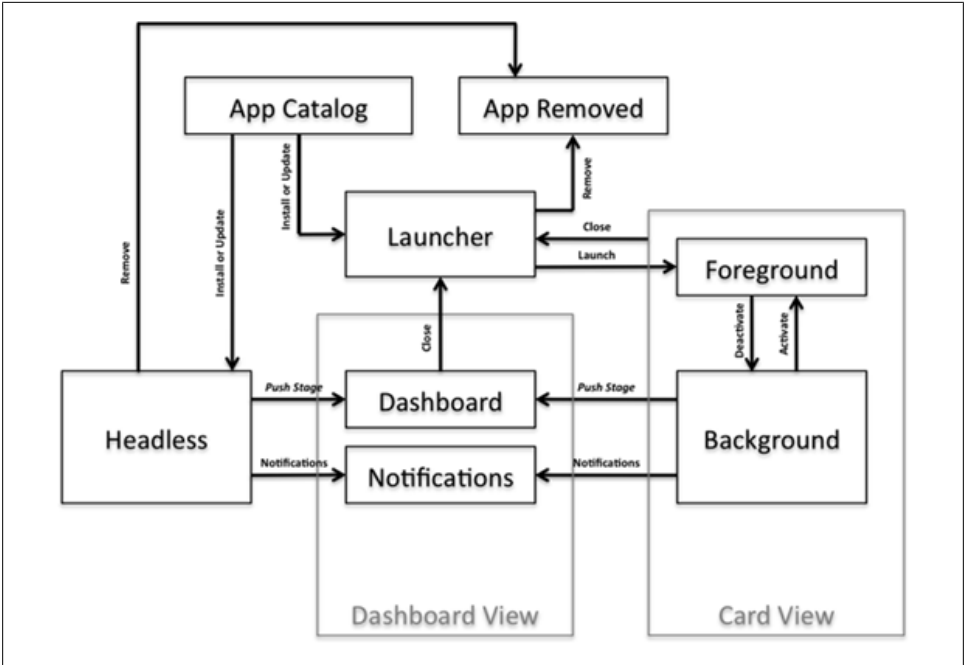


Figure 9. Application Stages

Downloading an application to the device initiates installation of the app provided that it has been validly signed. After installation the application will appear in the Launcher. If it is a *headless* application, then a card is not required and instead the application can utilize just a dashboard and communicate to the user through notifications. Headless applications typically include a simple card based preferences scene to initiate the application and configure its settings. Note that headless applications require at least one visible stage at all times (either a card, dashboard or alert) to not be shut down.

Other applications are launched from the launcher into the foreground and may be switched between foreground and background by the user. Each of these state changes (launch, deactivate, activate, close) is indicated by one or more events. Applications are able to post notifications and optionally maintain a dashboard while in the background.

Applications are updated periodically by the system. If running, the application is closed, the new version installed, and then it's launched. There isn't an update event so the app needs to reconcile changes after installation, including data migration or other compatibility needs.

The user can opt to remove an application and its data from the device. When the user attempts to delete an application, the system will stop the application if needed and remove its components from the device. This includes removing it from the launcher

and any local application data, plus any data added to the Palm application databases such as Contacts or Calendar data.

## Stages and Scenes

Palm's user experience architecture provides for a greater degree of application scope than is normally considered in a web application. To support this and specific functions of the framework, Palm has introduced a structure for webOS applications built around *stages* and *scenes*.

A *stage* is a declarative HTML structure similar to a conventional HTML window or browser tab. Palm webOS applications can have one or more stages, but typically the primary stage will correspond to the application's card. Other stages might include a dashboard, or other cards associated with different activities within the application. You should refer back to the earlier example of the Email application where the main card held the Email inbox and another card held a draft Email. Each email card is a separate stage, but part of the same application.

*Scenes* are mutually exclusive views of the application within a Stage. Most applications will provide a number of different kinds of scenes within the same stage, but some very simple applications (such as Calculator) will have just a single scene. An application must have at least one scene, supported by a controller, a JavaScript object referred to as a *scene assistant*, and a scene view, a segment of HTML representing the layout of the scene.

Most applications will have multiple scenes. You will need to specifically activate (or *push*) the current scene into the view and *pop* a scene when it's no longer needed. Typically, a new scene is pushed after a user action, such as a *tap* on a UI widget and an old scene is popped when the user gestures *back*.

As the terms imply, scenes are managed like a stack with new scenes pushed onto and off of the stack with the last scene on the stack visible in the view. Mojo manages the *scene stack* but you will need to direct the action through provided functions and respond to UI events that trigger scene transitions. Mojo has a number of stageController functions specifically designed to assist you, which you can find in detail in Chapter 2, Application Basics, and Chapter 3, UI Widgets.

## Application Lifecycle

Palm webOS applications are required to use directory and file structure conventions to enable the framework to run the applications without complex configuration files. At the top level the application must have an *appinfo.json* object, providing the framework with the essential information needed to install and load the app. In addition, all applications will have an *index.html* file, an *icon.png* for application's Launcher icon, and an *app* folder, which provides a directory structure for *assistants* and *views*.

By convention, if the app has images, other javascript or application-specific CSS, then these should be contained in folders named *images*, *javascripts*, and *stylesheets* respectively. This is not required but makes it simpler to understand the application's structure.

Launching a webOS application starts with loading the *index.html* file and any referenced stylesheets and javascript files, as would be done with any web application or web page. However, the framework intervenes after the loading operations and invokes the stage and scene assistants to perform the application's setup functions and to activate the first scene. From this point, the application would be driven either by user actions or dynamic data.

Significantly, this organizational model makes it possible for you to build an application that will manage multiple activities, that will be in different states (active, monitoring and background) at the same time.

Applications can range from the simple to the complex:

- Single scene apps, such as a Calculator, which the user can launch, interact with and then set aside or close;
- Headless apps, such as traffic alert application that only prompts with notifications when there is a traffic event and whose settings are controlled by its dashboard;
- Connected apps like a social-networking app, which provides a card for interaction or viewing and a dashboard giving status;
- Complex multi-stage apps like Email, which can have an Inbox card, one or more Compose cards, along with a dashboard showing email status. When all the cards are closed, Email will run headless to continue to sync email and post notifications as new emails arrive.

## Events

Palm webOS supports the standard DOM Level 2 event model. For DOM events, you can use conventional techniques to listen for any of the supported events and assign event handlers in either your HTML or JavaScript code.

The UI Widgets have a number of custom events, which are covered in more detail in Chapter 3. For these events you will need to use custom event functions provided within the framework. Mojo events works within the DOM event model but includes support for listening to and generating custom Mojo event types and is more strict with parameters; Mojo checks parameters to confirm that they are properly defined and typed.

The webOS Service functions work a bit differently, with registered callbacks instead of DOM-style events, and are covered starting in Chapter 7. The event-driven model isn't conventional to web development, but has been part of modern OS application design and derives from that.

## Storage

Mojo supports the HTML5 database functions directly and provides high-level functions to support simple Create, Read, Update or Delete (CRUD) operations on local databases. Through these Mojo Depot functions, you can create a local database and add, delete or retrieve records individually or as a set. It's expected that you'd use databases for storage of application preferences, or cache data for faster access on application launch or for use when the device is disconnected.

## UI Widgets

Supporting webOS's user interface are UI *Widgets* and a set of standard styles for use with the widgets and within your scenes. Mojo defines default styles for scenes and for each of the widgets. You get the styles simply by declaring and using the widgets, and you can also override the styles either collectively or individually with custom CSS.

The *List* is the most important widget in the framework. The webOS user experience was designed around a fast and powerful list widget, binding lists to dynamic data sources with instantaneous filtering and embedding objects within lists including other widgets, including other lists, icons and images.

There are *Simple Widgets*, including buttons, checkboxes, sliders, indicators, and containers. The *Text Field* widget includes text entry and editing functions, including selection, cut/copy/paste, and text filtering. A Text Field can be used singly or in groups or in conjunction with a List widget.

*Menu* widgets can be used within specified areas on the screen; at the top and bottoms are the *View* and *Command* menus and they are completely under your control. The *App Menu* is handled by the system, but you can provide functions to service the Help and Preferences items or add custom items to the menu. Each of the various menu types is shown in Figure 10.

For *Notifications*, you can choose from a Popup Notification or a Banner Notification, both of which post notifications for applications in the notification bar.

*Pickers* and *Viewers* are more complex widgets. Pickers are for browsing and filtering files or contacts, or for selecting addresses, dates or times. If you want to play or view content within your application, such as audio, pictures, video or web content, then you would include the appropriate viewer.

## Using Widgets

A widget is declared within your HTML as an empty `div` with an `x-mojo-element` attribute. For example, the following declares a Toggle Button widget:

```
<div x-mojo-element="ToggleButton" id="my-toggle"></div>
```



Figure 10. Application Menu Types

The `x-mojo-element` attribute specifies the widget class used to fill out the div when the HTML is added to the page. The `id` attribute is required to reference the widget from your Javascript and must be unique.

Typically, you would declare the widget within a scene's view file, then direct Mojo to instantiate the widget during the corresponding scene assistant setup method using the scene controller's `setupWidget` method:

```

/ Setup toggle widget and an observer for when it is changed.
// this.toggle      attributes for the toggle widget, specifying the 'value'
//                  property to be set to the toggle's boolean value
// this.togglemodel  model for toggle; includes 'value' property, and sets
//                  'disabled' to false meaning the toggle is selectable
//
// togglePressed    Event handler for any changes to 'value' property

this.controller.setupWidget('my-toggle',
  this.toggle = { property : 'value' },
  this.toggleModel = { value : true, disabled : false });

this.controller.listen('my-toggle', Mojo.Event.propertyChange,
  this.togglePressed.bindAsEventListener(this));

```

This code directs the scene controller to setup `my-toggle` passing a set of attributes, `toggle`, and a data model, `togglemodel`, to use when instantiating the widget and to

register the `togglePressed` function for the widget's `propertyChange` event. The widget will be instantiated whenever this scene is pushed onto the scene stack.

To override the default style for this widget, you would select `#my-checkbox` in your CSS and apply the desired styling (or use `.checkbox` to override the styling for all checkboxes in your app). For example, to override the default positioning of the toggle button to the right of its label so that it appears to the left of the label:

```
#my-toggle { float:left;
}
```

There's a lot more to come so you shouldn't expect to be able to use this to start working with any of these widgets at this point. Chapter 3 and 4 describe each of the widgets and styles in complete detail.

## Services

Even limiting yourself to just webOS's System UI, application model and UI widgets, developers would have some unique opportunities for building web applications, particularly with the notification and dashboards. But they'd be missing the access and integration that comes with a native OS platform. The Services functions complete the webOS platform, fulfilling its mission to bridge the web and native app worlds.

Through the Services APIs, you can access hardware features on webOS devices (such as location services, the phone, and the camera) and you can leverage the core application data and services that have always been a key part of a Palm OS device. Almost all of the core applications can be launched from within your application, and there are functions to access data in some core applications.

A service is an on-device server for any resource, data, or configuration that is exposed through the framework for use within an application. The service can be performed by the native OS (in the case of device services), an application, or by a server in the cloud. The model is very powerful as evidenced by the initial set of offered services.

The Services differ from the rest of the framework because they are called through a single controller function, `serviceRequest`. The request passes a JSON object specific to the called service and specifying callbacks for success and failure of the service request.

Starting with Chapter 7, you'll find a full description of the general model and handling techniques as well as enumeration of all the services and the specifics for using each one.

## Palm webOS Architecture

The Palm webOS is based on the Linux 2.6 kernel, with a combination of open source and Palm components providing user space services, referred to as the *Core OS*.

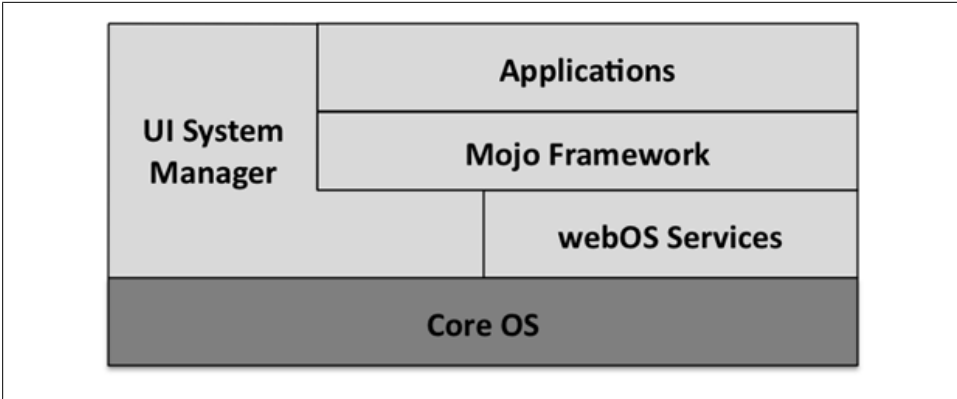


Figure 11. Simplified webOS Architecture

You won't have any direct interaction with the Core OS, nor will the end users. Instead your access is through Mojo and the various services. Users interact with the various applications and the UI System Manager, which is responsible for the System UI. Collectively this is known as the *Application Environment*. Figure 11 shows a simplified view of the webOS architecture.

This overview is included as background in case you want to have an idea of how webOS works—this information is not needed to build applications so you can skip it if you aren't interested.

## Application Environment

The application runtime environment is managed by the UI system manager, which also presents the System UI manipulated by the user. The framework provides access to the UI Widgets and the Palm Services. Supporting this environment is the Core OS environment, an embedded Linux OS with some custom sub-systems handling telephony, touch and keyboard input, power management, storage and audio routing. All these Core OS capabilities are managed by the Application Environment and exposed to the end user as System UI and to the developer through Mojo APIs.

Taking a deeper look at the webOS Architecture, Figure 12 describes the major components within the Application Environment and the Core OS.

The Application Environment refers to the System User Experience and the feature set that is exposed to the application developer, as represented by the Mojo Framework and the Palm Services. The Core OS covers everything else: from the Linux kernel and drivers, up through the OS Services, Middleware, Wireless and Media sub-systems. Let's take a brief look at how this all works together.

The UI System Manager or UI SysMgr, is responsible for almost everything in the system that is user visible. The application runtime is provided by the Application Manager, which loads the individual applications, and hosts the built-in framework and some

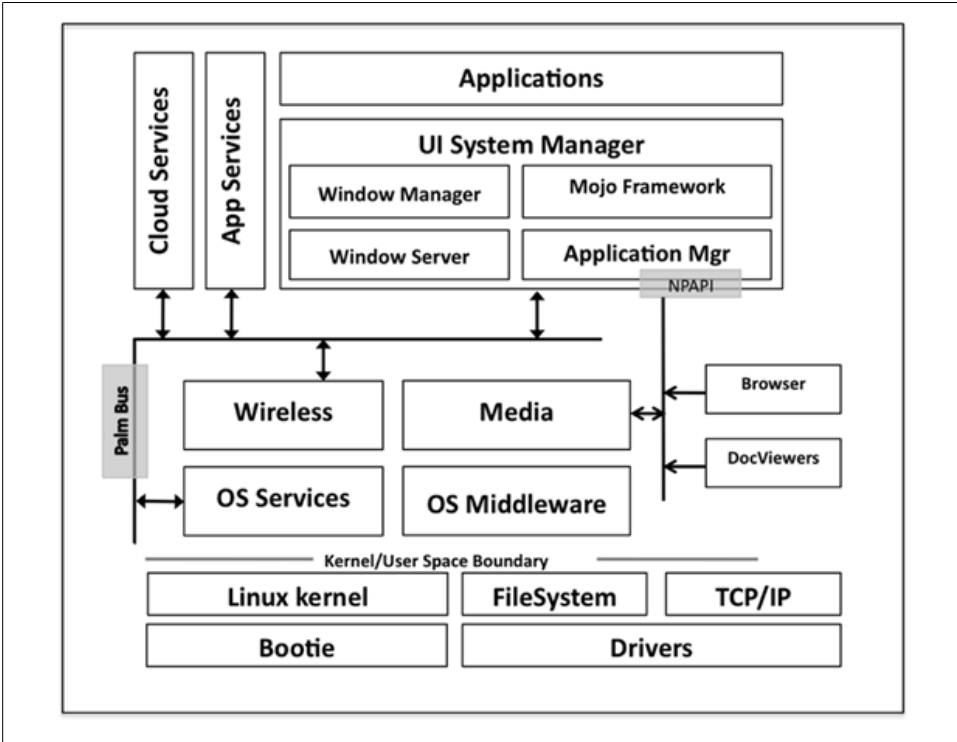


Figure 12. webOS System Architecture

special system apps, the status bar and the Launcher. The Application Manager runs in a single process, schedules and manages each of the running applications, and handles all rendering through interfaces to the Graphics sub-system and on-device storage through interfaces to SQLite.

Applications rely on the framework for their UI features set and for services access. The UI features are built into the framework and handled by the Application Manager directly but the service requests are routed over the Palm Bus to the appropriate service handler.

### Core OS

The core OS is based on a version of the Linux 2.6 kernel with the standard driver architecture managed by udev, with a proprietary boot loader. It supports an ext3 filesystem for the internal (private) file partitions and fat32 for the media file partition, which can be externally mounted via USB for transferring media files to and from the device.

The Wireless Comms system at the highest level provides connection management that automatically attaches to WAN and WiFi networks when available, switches connections dynamically, prioritizing WiFi connections when both are available. EVDO or UMTS telephony and WAN data is supported depending upon the particular device model. Palm webOS also supports most standard Bluetooth profiles and provides simple pairing services. The Bluetooth sub-system is tightly integrated with audio routing to dynamically handle audio paths based upon user preferences and peripheral availability.

The media server is based upon *gststreamer* and includes support for numerous audio and video codecs, all mainstream image formats, and supports image capture through the built-in camera. Video and audio capture is not supported in the initial webOS products, but is inherently supported by the architecture. Video and audio playback supports both file and stream-based playback.

## Software Developer Kit (SDK)

Of course the best way to get started writing webOS applications is to continue reading this book, but you should also go to Palm's developer site, <http://developer.palm.com> and register as a Palm developer and download the Palm Software Developer Kit (SDK). The SDK includes the development tools, sample code, the Mojo Framework, along with access to the Palm Developer Wiki, where developers will find formal and informal training materials, tutorial and reference documentation. Palm also provides registered developers with direct technical support by email or through interaction in a hosted developer forum.

## Development Tools

The Palm Developer Tools (PDT) are installed from the SDK and include targets for Linux, Windows (XP/Vista) and Mac OS X. The tools enable you to create a new Palm project using sample code and framework defaults, search reference documentation, debug your app in the weOS emulator or an attached Palm device, and publish an application. Chapter 2 includes more details about the tools in Palm's SDK and third-party tools, but you'll find a brief summary in Table 1 below.

Table 1. Palm Developer Tools

Tools	Major Features
SDK Bundle Installer	Installs all webOS tools & SDK for 3 <sup>rd</sup> party editors
Emulator	Desktop Emulator and Device Manager
Command-Line Tools	Create New Project Install & Launch in Desktop Emulator or Device Open Inspector/Debugger Window

Tools	Major Features
	Package & Sign App

---

The tools can be installed and accessed as command-line tools on every platform and include some bundles for integration into popular HTML editors and as a plug-in to Eclipse and Aptana Studio, a popular Javascript/HTML/CSS editor for Eclipse. Refer to Palm's Developer portal for the most current list of supported editors and tool bundles.

## Mojo Framework and Sample Code

The SDK installation includes a copy of the Framework and sample code to help you design and implement your application. Unlike most JavaScript frameworks, you won't need to include the Mojo framework with your application code since Palm includes the framework in every webOS device. The framework code included in the SDK is for reference purposes to help you with debugging your applications.

The sample code is also for reference. There are samples for most of the significant framework functions, including application lifecycle functions, UI widgets and each of the services. Simple applications are included to give you some starter applications to review and leverage as you choose.

## Developer Portal

Your main entry point is <http://developer.palm.com/>, which is where Palm hosts the Developer Portal. The portal provides access to everything that you might need to build webOS applications, including access to the SDK, all development tools, and documentation and training materials.

The Developer Portal provides your application signing services and access to the Application Catalog. This is an application store that is published and promoted with every webOS device through a built-in App Catalog application. Applications need to be signed for installation on a webOS device, and through the portal you can access the signing tools and related support.

## Summary

In this introductory chapter, you were introduced to webOS, Palm's next generation operating system. The following chapters will cover each of these topics in far more detail but this chapter should have helped you understand the webOS architecture and application model along with the basic services available in the SDK.

You'll find that it's pretty easy to get started writing webOS applications. After all, you're simply building web applications, using conventional web languages and tools. You

can port a very simple Ajax application by creating an *appinfo.json* file for your application at the same level as your application's *index.html* file. With as little as that, your app can be published and available for download to any webOS device.

From there you can invest more deeply by building in the Mojo UI widgets to take advantage of the fluid physics engine, gesture navigation, beautiful visual features, text editing, and the powerful notification system. You can move beyond simple foreground applications that rely on active user interaction, and adapt your application to run in the background or even be headless. Or consider an application that can open new windows for each new activity, enabling the user to multi-task within a single application. There's a whole new generation of applications possible on the webOS platform, just waiting to be built.