

- Overview
 - Who should read this document?
- Get to know the webOS platform
 - Take some apps for a test drive
 - Familiarize yourself with the technologies
- Palm webOS: A rich Software Development Environment
- Designing webOS Applications
 - Planning: it's the key
 - Start thinking like a "Mobile App Designer"
 - We know what it's like
 - Write your app's statement of purpose
 - Think about the world of possibilities
 - Interactions we support
 - Consider the technologies available to you
 - Sketch The Workflow
 - Sketching Your Scenes
 - Basic Elements
 - Scene Dimensions
 - Working On The Details
 - Start prototyping
 - Headers and Commands
 - Text and Buttons
 - Value Selectors
 - Lists
 - Visual Organizers
 - Connecting to Data and Services
 - The Application Menu
 - Providing Feedback
 - Progress
 - Progress Indicators
 - Activity Indicators
 - Errors
 - Notifications
 - Dashboard Applications
 - Layout and Customization
- Usability Testing
- Prepare for Delivery

Overview

The Palm® webOS™ software platform is built on webKit, which means that anyone who can design a website using HTML, CSS and Javascript can create a webOS application. You can also make use of advanced technologies, like AJAX and HTML 5 data storage. And to top it off, your app will be able to connect with and make use of the other applications running on the platform...quickly and easily.

Who should read this document?

This document is intended for designers and developers who want to design a webOS application. It contains:

1. An introduction to the webOS platform
 1. Describes what webOS environment and its applications looks like, how they work, and how users can interact with them
2. Palm's webOS Application Design Philosophy & Guidelines
3. Examples of "Good Design"
4. Details about UI Controls you can use in your application

If you need information about the webOS platform, its technologies and architecture, read [Palm webOS Overview](#).

Get to know the webOS™ platform

The best way to gain an understanding of how webOS applications work (and how they work together) is to *use* them...but we realize that you may not have a device yet. It's easy (and free) to [download the Palm® Mojo™ SDK](#), which includes the webOS Emulator and try the apps and the OS yourself. It runs on Windows, Mac and Linux platforms. Using the applications will help you understand how the webOS platform delivers an outstanding user experience, and will give you ideas that you can use in your own applications. While using the apps, notice:

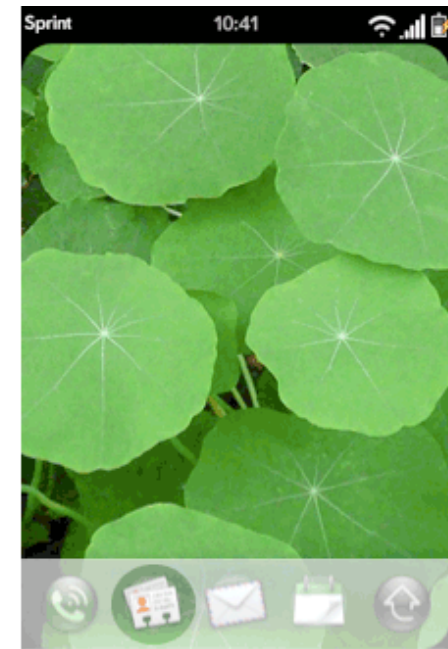
- How easy it is to interact with the device, using your finger to perform simple, natural gestures
- How you can run multiple applications at once, and quickly switch between them.
- How applications work together, to simplify your workflow
- How you can quickly search for any data, on your phone or on the web
- How applications notify you when important things happen

Take some apps for a test drive

Once you've installed the Emulator or have a webOS device, follow the instructions below to try a few of the applications. Screenshots are provided to help you see how the system works.

■ Launch & Switch Between Applications

- Tap the **Contacts** icon in the Quick Launch bar.
 - Use it to create a new Contact.
 - Edit the phone number, address, email address and IM address.
 - Exit the contact using the "Back" gesture (right to left stroke, under the screen, but above the Center button).
 - Then press the Center button. The Contacts app minimizes to a "Card."
- Tap the **Calendar** icon in the Quick Launch bar.
 - Create a new appointment by tapping on a date and typing the event name.
 - Tap the "i" to set details for the event.
 - Press the Center button again. The Calendar app minimizes to a card.
- Tap the **Email** icon in the Quick Launch bar.
 - Input your account settings and connect to your email service.
 - Setup an account for another service, as well (if you have a second email account).
 - The email app fetches your email, and allows you to browse your mail account-by-account, or all emails together in a single list.
 - Press the Center button again to minimize the application.
- Now flick left-to-right over the cards, to scroll through them. Tap the one in the center to bring it forward.
 - The webOS operating system is a multitasking environment. Even when an app is not maximized (occupying the whole screen), it's still working...in a card, in the background.



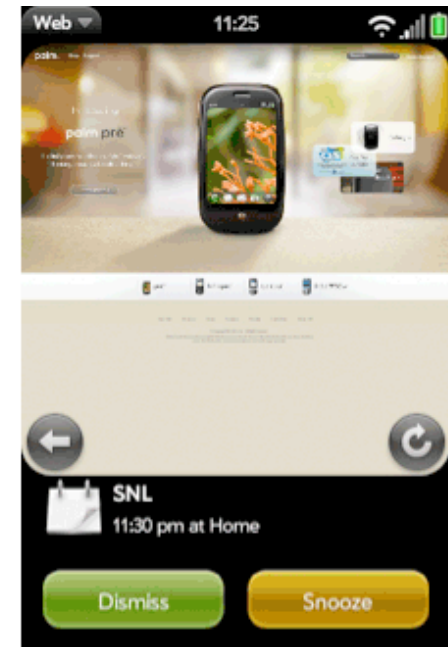
1. This is the QuickLaunch Bar.
Tap "Contacts"

■ Polite Notifications

- Imagine that you're browsing a web page, and you get an email or a text message. The webOS platform allows any running application to display a small notification at the bottom of the screen. These notifications politely let you know that what happened, without interrupting your work and insisting that you tap them before you can return to whatever you might have *actually* been doing.

■ Urgent Notifications

- Imagine that you're writing an email and you've lost track of time...and it's time to leave for an important meeting. The webOS platform allows any application to display a Popup Notification, to inform you of the *really important thing* it needs to tell you. In the example, the Calendar application would have displayed a Popup Notification to let you know that it's almost time for the meeting. Popup Notifications slide up from the bottom of the screen. As a developer, you can display anything in them. The Calendar app uses them to remind users of upcoming events. Users can "Dismiss" the reminder (which causes it to disappear) or "Snooze" it (to minimize the notification to a Dashboard Summary icon, so the user can still see the reminder while they continue working with the current application).



7. Dashboard Notification from the Calendar app appears. User taps Dismiss.

Familiarize yourself with the technologies

Palm webOS hardware devices contain many sensors which you can take advantage of in your applications. Here's a quick overview of each one.

■ Touchscreen

- Palm webOS devices have a capacitive touchscreen, which allows users to *touch* the screen to interact with applications...instead of having to use other things, like pens or 5-way controllers. The user can:
 - **Tap** buttons, menu or list items
 - **Double-tap** to perform actions like zoom in/out
 - **Flick** to scroll through lists, or quit an application
 - **Swipe** to the right in lists, to delete items
 - **Touch & Drag**, to scroll
 - **Touch & Hold**, to enter "reorder" mode. Then drag items to move them.
 - **Spread/Pinch** using 2 fingers, to zoom in/out on a web page or a picture
 - **Touch** with 2 fingers and **rotate** - to rotate content

■ Gesture Area

- Palm webOS devices have an area below the Touchscreen, where users can use various gestures to accomplish certain tasks.
 - One of the most important gestures is called the "Back" gesture. It is a short, quick swipe from right to left, over the center button.

■ Sensors

■ Orientation

- Palm webOS devices contain sensors that report its orientation. When a user moves, rotates or tilts the phone, the device reports the change, and your application can respond to it. Here are some simple examples:
 - Launch the **Web** application. While holding the phone up vertically, type in a URL and view the page. Now turn the phone 90 degrees, to widescreen orientation. The Web app senses the event and redraws its content in widescreen format.
 - Launch the **Contacts** application. While holding the phone vertically, create a contact and, while editing it, turn the phone 90 degrees to the right. The Contacts app ignores the rotation because there's no particular value to the user to draw the UI in widescreen format.
- When developing your own application, consider whether there is value in responding to a rotation or orientation change. If so, you can take advantage of any of that data.
 - Currently, the webOS platform reports the following rotation and orientation events: 0, 90, 180, and 270 degrees, & faceUp, faceDown.
 - As development continues, a complete set of positional events and data will become available to you, so your app can take advantage of motion events in all directions.
 - This data will be particularly useful for game developers, who can use device rotation/orientation values to allow users to "navigate" through three-dimensional gaming environments.

■ Proximity Sensor

- A Palm webOS device knows when something comes close to it, and reports that event via the OS, so an application can respond to it.
 - The Phone app uses the Proximity Sensor value to turn the screen off when a person holds the phone close to their face. This is done to prevent the user from *accidentally* pressing buttons with their cheek while talking on the phone.

■ Location services

- Palm webOS devices contain a GPS radio, which uses satellite data to report the current time and determine the current location. Your app can make use time & location data to serve your users better.
 - An application that find movies in your area, in the next 3 hour time window.
 - A map application that can display the traffic near your current location
 - A picture application that automatically adds the time, date and location where it was taken

■ webOS Platform APIs

- The webOS development environment provides you with API's that allow you to create UI using standard controls, use JavaScript to make that UI do powerful things (like access hardware services, launch other webOS applications and make use of their data). You can see an example of how this works in the Contacts app. You can tap a contact's phone number to call them using the phone app. Tap their address to display it in the Maps app, and so on.

Palm webOS: A rich Software Development Environment

One of the major goals of the webOS platform was to make it possible for you to create great mobile applications that deliver the richness of a desktop user experience on a mobile device. And we did that by:

■ Building an OS on industry-standard technologies

- The Palm® webOS™ operating system is built on webKit, the code which is underneath many of today's best web browsers. It's solid, open-source and provides you with a great foundation.

■ Extending the OS, so you can do more in your applications

- Palm webOS operating system connects to all of the amazing hardware technologies that webOS devices have...and makes them available

to you, so your app can take advantage of them.

- The webOS development platform contains the Mojo Framework classes, which provide the following:
 - A comprehensive set of desktop-class UI controls. Including controls you can't find on other platforms, such as:
 - A customizable application menu
 - Text edit commands like Cut, Copy, and Paste
 - Popup menus and more
 - Cascading Style Sheets (CSS) you can use to quickly customize the look of your application.
 - Applications like Email and Contacts use the webOS platform's default style, whereas apps like Music Player and Camera use the "Dark" style. You can use either of these, or create one of your own, to create your own look.
- **Using a "development language" you already know**
 - HTML/DHTML & CSS
 - To render the "pages" in your application, and make them look wonderful.
 - Javascript
 - To make the buttons, menus, lists, etc in your application do something useful.
 - HTML 5 DB
 - To store data locally on the device.
 - AJAX

The goal was to create an application development environment where **anyone** - from the first-time website developer to the most-seasoned application developer - could design, implement and deliver a mobile application.

Designing webOS Applications

At its core, the webOS operating system and its applications were designed to allow users to interact naturally and simply with their phone. Everything works together seamlessly. It's simple for users to manage the various aspects of their lives - from personal items to work items, tasks to entertainment, and the list goes on and on. The goal was to ensure that their phone feels like the best personal assistant they could have ever wished for, because it helps them stay on top of the things that matter - appointments, communications with others, errands...and even provides ways for them to relax and enjoy themselves with a little music, movie-watching or web-browsing. The goal was to create a device that becomes truly indispensable, something that a user couldn't have ever even imagined living without.

Palm has delivered a set of applications that make use of all of technologies that were described in the previous section...and has put a lot of care and effort while doing so. Previously in this document, we encouraged you to watch demos and try the applications so you'd understand how they work, and how they make use of the technologies the webOS platform and hardware technologies make available to you, so you'll have a context for understanding the User Interface Philosophy and Design Guidelines that will be discussed next.

Planning: it's the key

Mobile devices and the applications that run on them are typically used by very busy people, who usually have only short bursts of time in which to use your application. For this reason (and many others), it's imperative that your application design is clear, simple and helps the user succeed without ever feeling like the application is "in their way." The not-so-subtle subtext of this message is that you really have to *think* about your design upfront. Here are some tips that will help you through that process.

- **Think about UI Design the "Palm Way"**

At Palm, we believe that users deserve applications that do something very specific, and do it well. Think about *the most important things* a user

would do with your app...and create a simple "statement of purpose" that describes it. Use that statement to determine which features to include...and which ones not to. Here's an example:

- We know that many people have multiple email accounts, so we designed the Email app to allow users to connect to multiple accounts, view their email from each of them...together or separately...and to read, compose, reply, and manage their email. This is actually a very tall order, but notice that Email's mission statement says nothing about managing contacts or anything else. It's an email manager, and that's all. The Email app makes use of other apps and data stores on the Pre, like Contacts. Your app should intelligently leverage other applications and their data, too.
- **Design an app that works like other apps users already know**
Users will learn how the Pre works by using its built-in applications. If your application works like they do and uses standard webOS UI controls, users will know how to use your app right away.
- **Design a beautiful app**
The emotional impact of an attractive application cannot be underestimated...and it's easy to achieve! The UI Framework provides you with:
 - Standard UI controls you will use in your application.
 - Default styling for all controls, so your app looks great from the start.
 - Start with the default controls. If you want to customize the look of your app, you can...and we'll talk more about that later.
- **Design an app that is delightful to use**
Applications delight users when they enable them to get their tasks done quickly and efficiently. This is especially important for mobile users, who often only have 10 or 15 seconds to use your application. Your app will be their favorite if it helps them get something important done quickly, efficiently and easily.

Start thinking like a "Mobile App Designer"

The Palm Pre display is 320 pixels wide x 480 pixels high. You won't be able to fit **every screen** and **every feature** from your website, desktop app or mobile app in your webOS app. Heed the good advice given long, long ago: ["Don't try to fit a mountain into a teacup"](#). It was good advice then, and it still is now! When determining what to include in your product, focus on what *users really need* when they're on the go, and use your well-crafted statement of purpose to determine the features that *really* need to be in your app. We'll help you see how this process works later in the document.

We know what it's like

The people at Palm have been in the Mobile UI business for a long time...and some have come from other backgrounds, as well (web design, desktop application design and feature phone app design). We know what it takes to design and deliver a *great* mobile application. Here are few tips to help you.

- **Are you a website developer?**
 - Good! Because you are already familiar with HTML, CSS, and Javascript, you're halfway down the road to success. The next step is to think about the product you want to create (or how you can use the best things from your website) and deliver those in an outstanding mobile application that will run on (and take advantage of) the webOS platform.
- **Are you a Desktop Application Designer?**
 - Even though many mobile devices have screens smaller than what you're used to designing for, we think you'll feel right at home designing a webOS application. The webOS platform has many of the same desktop UI controls you're familiar with. Your biggest challenge will be figuring out which of your desktop features users truly need when they're on-the-go, and discovering what new features you might want to add, given the great technologies that the webOS platform provides. Focus on the features that are most important to your users when they're on-the-go, deliver them in a nice, simple workflow and your customers will be thrilled.
- **Are you a Mobile App Developer?**
 - Porting an app from a **Feature Phone?**

- We know what that's like and we think you'll love designing apps on the webOS platform. Palm's webOS development environment is much richer, and the device's screens are considerably bigger than most feature phones. This allows you to display more content and UI controls than you ever could in a feature phone UI. Think about it - a richer UI + more content area = fewer screens to develop and manage. And you won't have to jam all of your commands in a single menu anymore.
- Porting an app from a **competing Mobile OS?**
 - Some mobile OS' are very sophisticated, but are also limiting in many ways. Take our advice when we say "Don't just port your workflow" straight to the Pre. Instead, take a moment to notice the kinds of controls that the webOS platform provides you with. We have *actual* popup menus...that are displayed in the **same scene** you are currently in. You don't HAVE to open a different scene in order to choose something from a list...or edit a value. Seriously. Take a moment and get to know the core applications on the Pre, notice how they're different and better than what you're used to...and take this opportunity to simplify your application's workflow. You'll realize very quickly how much simpler your app can be, once you've updated your design for the webOS platform.

Write your app's statement of purpose

It's a lot of fun to brainstorm and come up with great ideas for an application...but designing and developing a great *mobile* application requires you to whittle that huge set of ideas down into a small manageable set, consisting of the most important and most useful features. Go ahead and brainstorm - we heartily encourage it - and then take a good, hard look at those ideas, and determine which features *really* need to be in your webOS application. Guide your thinking by working through these questions:

- **Who is this product for?**
 - What kind of person might want or need it?
 - How busy are they, and where/when might they use it?
- **Which features really *define* this product?**
 - Is it a productivity application? If so, what's the most important thing it will help someone do?
 - Is it an information application? What kind of information will it deliver, and what's the best way to deliver it to people?
 - Is it an entertainment application? What's the simplest and most delightful way that entertainment can be delivered?
- **Which features would make your app indispensable?**
 - Discard any features that prevent (or distract) you from achieving this goal
- **Is your app an "interactive app" or does it simply "provide a service?"**
 - Most applications are interactive, and require you to create a user interface. The bulk of this document is devoted to explaining how to design a great UI.
 - Some applications don't really have a full UI. They simply provide users with information via notifications. We call these "Dashboard" applications.
 - If you want to develop a Dashboard app, read the section on Notifications, as this will be the most helpful areas for you. Dashboard apps must have an application icon (so users can start them) and at least one visible UI element. It doesn't have to be a card. It can be as simple as a Dashboard Summary Icon and related Dashboard Item UI.

This document is intended to help you design an outstanding UI for an interactive application, so let's look at an example. Palm includes an Email application on webOS devices, and it's a very useful application...and it's useful because it does what it should do, and nothing else. Here's Email 1.0's statement of purpose:

Email 1.0 enables users to:

- connect to personal and corporate mail servers.

- read email from multiple accounts together or separately. Open attachments.
- compose an email message and send it from any account. Include attachments.
- reply to the sender, all recipients, forward, delete and manage any email.

Notice that all of these statements focus on one thing: email, and allowing the user to interact with all of the email that they receive. It says nothing about "being the application that opens the Excel or PDF files" that are attached to emails, or about managing all of the Contacts that you might send email to or receive it from. The team that wrote the Email program knew that they only had to do one thing: effectively manage email, and that they could make use of the other services and applications in the webOS environment to do things like manage Contacts and open files.

Think about the world of possibilities

The webOS platform's user interaction model is based on the idea that everything is "live" and can be interacted with using a very natural pointing device - your finger. This opens up a world of possibilities, and also means that everything on the screen must be big enough to tap...while sitting still, while walking, riding in a car, and so on. The **point** is that your UI must be designed with all of these things in mind. And, in order to keep interactions simple, any changes a user makes are made in realtime. There's no need for a SAVE button or command in any application. Keep this in mind as you design and implement your application.

Interactions we support

- **On the Touchscreen**
 - Tap - invokes an action or opens an item
 - Double Tap - to zoom in, like in the Web app
 - Flick - to scroll up/down, left/right
 - Tap & Hold, then Drag to move - like moving applications in the Launcher
 - Pinch/Spread - to zoom in/out of content, like in the Web app
- **In the Gesture Area**
 - Back right-to-left gesture. Exits the current scene.
 - Press the Gesture area & Tap an application icon
 - To perform "meta" actions on an app, like "Get Info"
 - Press the Gesture area + a letter to perform Edit menu keyboard shortcuts:
 - Cut (X), Copy (C) and Paste (V)
- **Text Fields**
 - Tap - to position the insertion cursor
 - Orange + moveFingerAnywhereOnscreen - to move the cursor
 - Shift + moveFingerAnyDirection - to select text in either direction
- **Card view**
 - Flick upward to "throw the card away" - quits the application.
 - Press & drag - to reorder cards.
- **Lists**
 - Press and and slide left or right to delete (alternately with confirmation)
 - Press & Drag up or down to reorder items
 - Press & Hold on list title to allow a user to edit the name
- **On the keyboard**
 - Text Entry

- Global Search
- Keyboard Shortcuts
 - Press the Gesture Area + shortcut letter

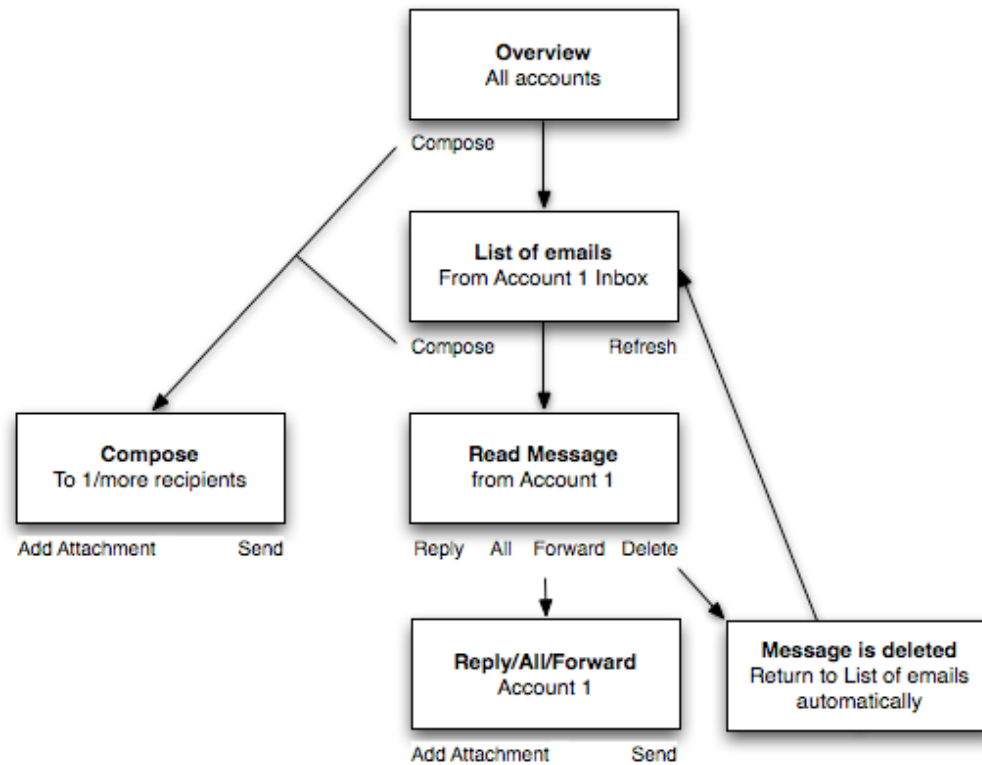
Consider the technologies available to you

Think about the Palm webOS operating system and the hardware technologies you can use to make a truly outstanding application.

- Direct connectivity to other Palm webOS applications
 - So you can access their data, using it when you need to...rather than having to create/manage that data yourself.
 - Contacts, Calendar, Time/Date, Web Browser, Maps, and more.
- Built-in network connectivity (via wi-fi and/or cellular connections)
- Location Services (GPS)
- Accelerometer (Rotation), Proximity Sensor, Light Sensors

Sketch The Workflow

The first thing you should do when designing a mobile application is to "work out the workflow"...so put that mouse down and grab a pencil and piece of paper...or grab a dry-erase marker and start drawing on a whiteboard. Draw simple boxes that represent each screen (or scene, as we call them) in your design. Connect them with arrows, so you know what will happen when the user taps an item on the screen...and then take a step back and *really* examine what you designed. Have you created a workflow that helps the user accomplish the most important things they'll do with your app, using the fewest steps? Remember that they may only have 15 seconds at a time to use your application. That's a humbling thought, isn't it? Make sure the workflow's smooth and efficient. Let's take a look at Email 1.0 as an example.



Email 1.0's workflow accomplishes the user's primary goals, which are to:

- Read emails quickly
- Create email messages quickly
- Reply/forward emails quickly

These are actually **performance goals**, which nicely complement the Email product's mission statement. Performance goals keep you honest, and help you focus on efficiency. When designing your application, think about the tasks that users need to do **most often**, and design your workflow so your users can do those things quickly and efficiently.

Sketching Your Scenes

Basic Elements

In Palm webOS applications, we call each step in the workflow a "scene." A scene is analogous to a window or dialog box in a desktop application, or a page in a website design. A scene commonly contains the following elements:

- **Header** (Optional)
 - Displays a Fixed or Scrolling header, which is used to display the title of the scene, and any other controls that affect the view. There are 2

kinds:

- A Fixed Header remains in place, even when the user scrolls content underneath it. Fixed headers are 50 px tall.
- A Scrolling Header scrolls out of view when the user scrolls content underneath it. Its height can vary. The example below shows a scrolling header that is 68 px tall.

■ Scroll View

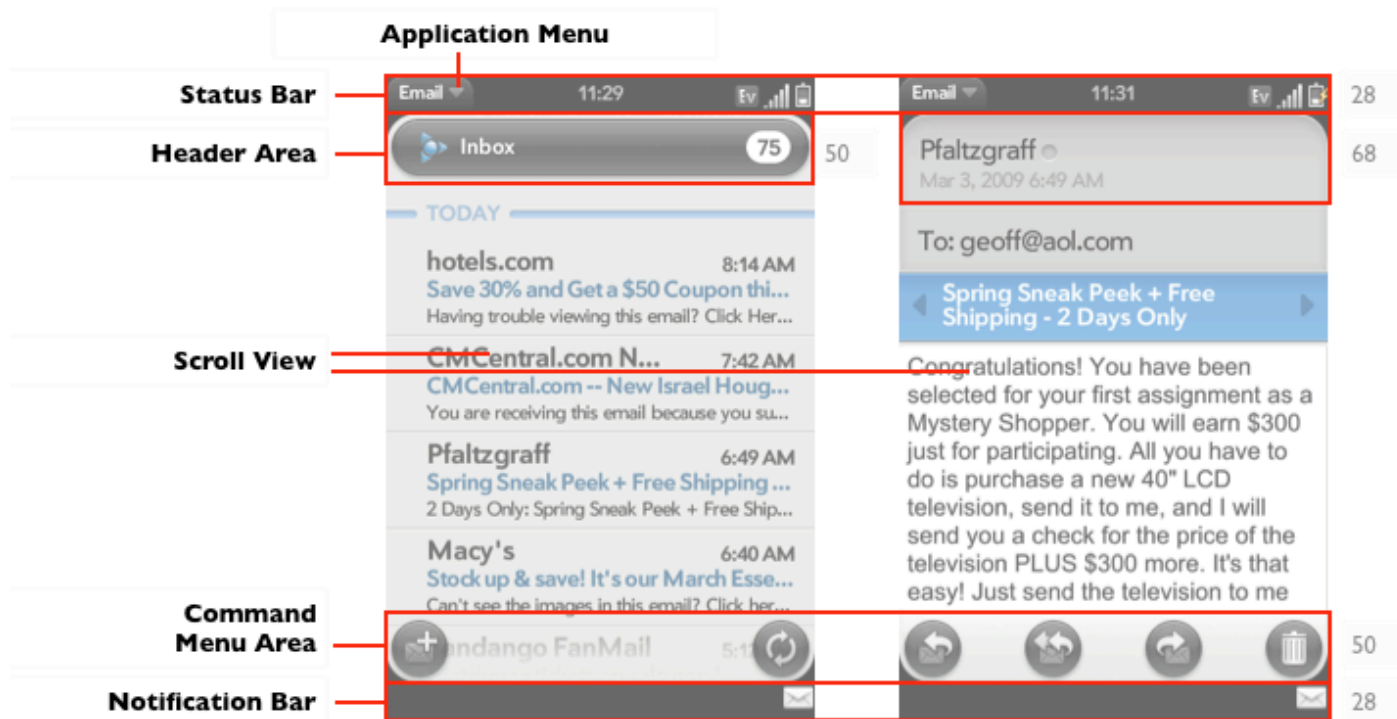
- The area that displays the UI Controls or information needed in the scene.
 - Similar to content in a web page or web application.
 - The user can scroll content in this area up and down freely.

■ Command Menu (Optional)

- Displays Command Buttons, which are used to perform the **most important actions** in the scene.
 - Buttons appear at the bottom of the scene, in the Command Menu area.
 - They float above the content in the Scroll View, when the user scrolls it. This means the buttons will always be visible.

■ Application Menu commands

- Appears in the upper left hand corner of the display, in the Status Bar
- Is used to display commands for the "other actions" a user may need to perform in a scene.
- You can insert new commands in the App Menu in each scene.



Scene Dimensions

Each webOS device has a specific screen size...and certain OS-based elements will occupy a fixed amount of space on that screen. Many developers ask "How much space will I have to design with in a scene" and the answer is "It depends." To answer the question, you need to know which items the OS will display, and how tall each area is. There are also elements that are commonly included in scenes that require a fixed amount of space. Let's use the Palm Pre as an example. Its screen is 320 pixels wide by 480 pixels high (portrait mode) and 480 x 320 (landscape mode).

OS-Controlled Areas

- The **Status Bar** appears at the top of the screen and is **28** pixels high. It contains the Application Menu, the Time and the Connection menu. It occupies 28 pixels of height and is always displayed.
- The **Notification Bar** appears at the bottom of the screen when a background application posts a notification. When it appears, it pushes the bottom of your application upward by **28** pixels.
 - Apps like Email or Messaging display banner notifications and dashboard summary icons in this area regularly. It's a good idea to "plan" that something will appear in this space when you're designing your app.
- **Popup Notifications** can also appear, and they vary in height. They can be as large as half the screen height, depending on what they're informing the user about.
 - Users typically see a popup notification and interact with it immediately...and it either disappears completely, or minimizes to the Notification Bar, occupying only 28 pixels of height
 - It's not necessary to worry about the height of Popup Notifications when designing your scene. They overlay your scene for a moment, and disappear quickly.

Common Scene Elements

- Scenes commonly contain a Fixed Header or a Scrolling Header, to serve as a title and contain other controls (if needed)
 - **Scrolling Headers** can be almost any height. We **suggest 50** pixels, but you can set them to be **any size**. Scrolling headers scroll out of view when a user scrolls the scene.
 - **Fixed Headers** stay in one location, and have a fixed height (**50** pixels). They float above content from the scene when the user scrolls up and down.
 - **View Menus** consist of a Fixed Header and a popup menu that can be used to control what is displayed in the scene below. They are 50 pixels tall and float above the scene, as well.
- Scenes commonly contain **Command Buttons** in the Command Menu Area. This area is **50** pixels high and can contain up to 5 items. They float above content from the scene when the user scrolls up and down.

As you can see, the answer to the question "how much space do I have in a scene" is not a simple one. It really depends on the screen size, what the OS displays, and what you have chosen to include in your scene. Here's an example to see how it would work on a scene on the Palm Pre. If a scene included a fixed header (**50** pixels) and we wanted to account for the appearance of notifications (**28** pixels) + the always-present Status Bar (**28** pixels), we have to assume that **106** pixels would be taken up by these things. On the Palm Pre, we'll be able to display the following amount of content in a scene: **320 wide x 374 high** in Portrait mode and **480 wide x 214 high** in Landscape mode (if the app supports rotation).

Some apps are different

We understand that some application designers and developers want to create a "clean, immersive experience, free of any unnecessary OS clutter." This is common desire among people who develop Games and Media applications. The webOS platform allows developers to run their app in "Full Screen Mode" for such occasions. This allows the application to take over the entire screen, hiding the Status and Notification Bars. The only notifications that are NOT

suppressed are Popup Notifications...which is a good thing. The OS allows those notifications to come through so won't miss an important phone call, meeting reminder.

Working On The Details

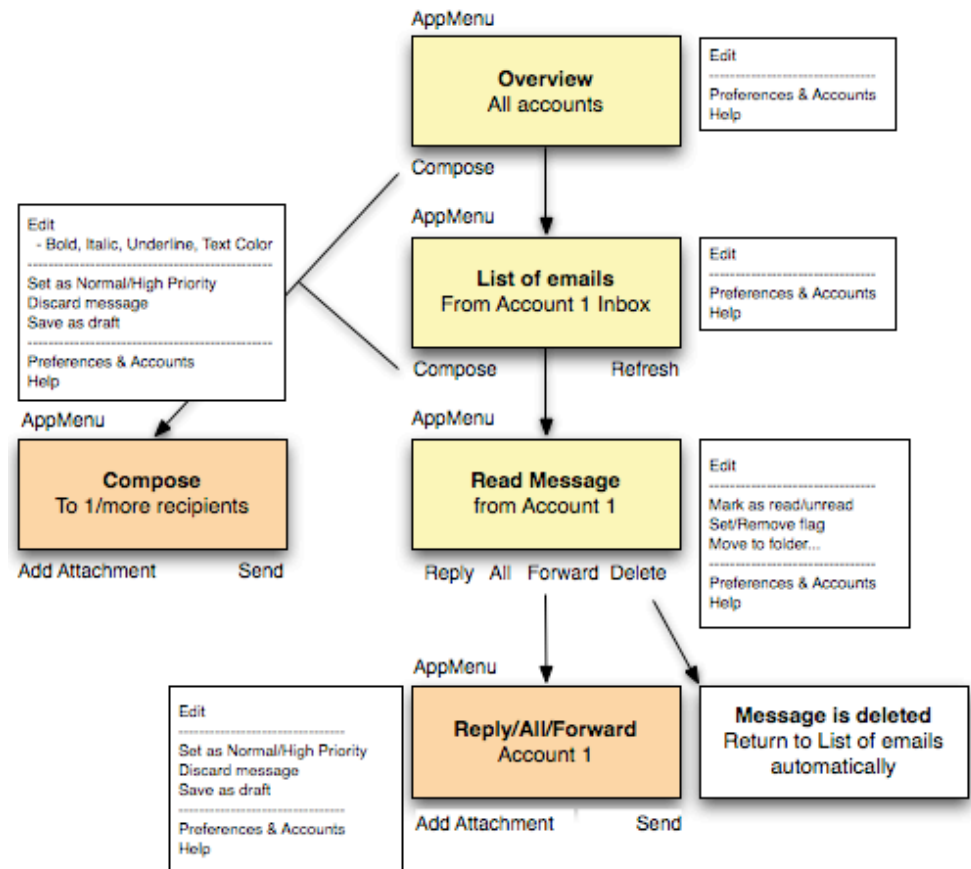
Now that you've got a pretty good idea about what you need to include in the workflow and how much space you have to work with, you should take a moment to consider the following things:

- **How many scenes do you really need?**
- **What buttons should you display at the bottom of each scene, and which ones should you put in the Application menu?**
 - The "back" gesture is a "half swipe" in the gesture area, that takes the user from the current scene to the previous one. Users will learn how to do it in the tutorial they see when they use their device for the first time. Every webOS application makes use of it.
- **When should a user action display the result in the current view vs. a new card?**
 - When your application launches, it's launched in a card...and, for the most part, you should display all of your content there. If certain tasks or actions in your app would "work better" or "more conveniently" if they were able to work in a new card, you should open one. We'll discuss this further in the Email app example.
- **What should happen when the user performs the "back" gesture on the gesture strip?**
 - The "back" gesture is a "half swipe" in the gesture area, performed over the Center button. It's what user will do to navigate from the current scene to the previous scene. Users will learn how to do it in the tutorial they see when they use their device for the first time, and every application will make use of it.

These are all very important questions, and are exactly the kind that affect how many scenes you need to design and develop, and how navigation works in your application. The team that wrote the Email 1.0 application had to think about all these things, too. Let's look at a portion of their application to see how they did it:

When Email 1.0 launches, it opens a **card** that displays the **Overview** scene. Let's take a moment to get an understanding of how **navigation** works within the application.

- When Email 1.0 launches, a new card is opened and the **Overview** scene appears. It displays a list of the Email Accounts the user has configured the app to display.
- The user taps an Inbox. The **List of Emails** scene is opened (in the same card), displaying the list of email messages in that Inbox.
- The user taps a message. The **Read Message** scene opens (in the same card).
 - If the user performed the Back Gesture now, they'd return to the **List of Emails** scene. Another Back Gesture would lead back to the **Overview** scene. Because each scene was opened in the same card, the Back Gesture navigates to the previous scene within that card. But let's not worry about that right now. Let's assume the user tapped the "Reply" button instead.
- The user taps the **"Reply"** button at the bottom of the **Read Message** scene. A **new card** is opened, where the user can reply to the message.
 - The **Reply** scene is actually the **Compose** scene, with the appropriate content filled in. And because Compose, Reply, Reply All and Forward require the same functionality, it's possible to use the same scene for each action.



Specify Buttons & Menu Commands

Each scene should have its own set of **command buttons** (represented as verbs in the diagram) and special commands in the **Application Menu** (represented in the white boxes near each scene). Sketch these out when working through your initial navigation flow. It will help you determine how many scenes you need, which actions are needed in each and how/when you can re-use a scene for a common purpose.

Open New Cards Only When Needed

In the example above, the Email team chose to open a new card when the user wanted to compose a new message, reply to one or forward one. You might be wondering why they chose to do that. Here's why they did it:

There are times when a user starts a task (like composing an email) and then realizes there's something they need to do before they can finish it. They might need to check information in a different email before writing their reply...or open the Web application and look for a URL they wanted to copy/paste into the message they're composing. This is exactly why we have the concept of "cards" in the webOS platform. So application designers and developers can support the user's workflow when they have needs like this. In a case like this, opening a new card is the right thing to do because it allows the user

to break away from their task for a moment without losing any of their work (as can the case in applications on other mobile OS platforms). When designing your scenes and your workflow, think about whether users might need to have a certain scene displayed in a new card. If there are good reasons to do so, do it. However, launch new cards only when necessary. When your app launches extra cards, it complicates the user experience because it adds to the set of cards displayed when the user presses the Center button. Each card also requires additional device resources. The guideline is "if there's a good reason to open a card and it purposefully supports the user's workflow, open one. Otherwise, open new scenes in the current card."

In many cases, applications can run perfectly in a **single card**. The Calendar app is a good example. Every action a user performs opens a scene in the same card...and the Back Gesture always returns them to the scene they had seen before. There's very little workflow-advantage to opening a new card in the Calendar app. The team that designed the Calendar app considered cases where there was *some* value in opening a new card, but decided against it because the value was not greater than the cost of the system resources required and the complexity added to the overall user experience. For those reasons, they decided that the Calendar app would open each new scene in the same card.

Use Animations Effects Consistently

The Palm applications that ship on webOS devices use animation effects when transitioning from scene-to-scene and performing other actions. Here's how to use animations properly in your application:

- **Moving within the application's hierarchy of scenes**
 - When moving down into the hierarchy, use the **Zoom In** effect
 - When moving back up the hierarchy, use the **Zoom Out** effect
 - When using the Back gesture, use the **Zoom Out** effect
 - **Example:** Launch the Email app. Tap one account, then one message item, then read the message - the application uses the **Zoom In** effect in each case. The Back gesture uses the **Zoom Out** effect when the gesture is performed.
- **Moving to a similar level in the hierarchy**
 - Use the Cross-Fade effect
 - Example: Launch the Phone application. Tap the Calls button (lower right hand corner). Tap All Calls, then Missed Calls. Both scenes exist at the same level in the application hierarchy and use the Cross-fade effect.
- **Launching another application from yours**
 - Example: Launch the Web app, tap a mailto link. This opens the Email application. This opens a new card and uses the **New Card** animation.

Figure Out When You Should "Save" Data

It is your app's responsibility to determine when changes users make to their data are saved. Whenever possible, you should save changes automatically. This simplifies the user experience in several ways: it removes the need for a "save" button in your application, and frees the user from ever having to worry about "losing their data" because they "forgot to save it." When designing your app, make sure you autoSave in these cases:

- The user edits a value in one field and taps into another
- The user edits a value and then performs the Back Gesture
- The user edits a value, minimizes the app to a card and then quits the app

There are some cases where it's better to allow the user to explicitly commit changes they've made in a scene. Here's an example. Your application requires a userid and password to login, and requires a user to reset the password every 2 months, it would be better to provide two buttons: one that commits the change and another that discards it. If you automatically updated the password and the user had made a typing mistake, they wouldn't be

able to access their account or data again. If the potential consequences of autoSave outweigh the benefits, allow the user to save the changes themselves.

Plan for Interruptions

A user might be using your app and, all of a sudden, an important meeting notification or text message arrives and they have to deal with it. The webOS platform handles this nicely by displaying notifications below your app in the Notification Bar...but your app should be ready, too. If you're creating the kind of app where interruptions could be a problem (e.g. a game), consider building autoPause functionality into your game when a Popup Notification arrives...or when a user quickly minimizes your app to a card because their boss just popped into their office!

Use System Resources Responsibly

One of the coolest things about the webOS platform is that your app can be running while others are, too. When your application is being used, your app should use system resources in the best way it can. If the user (or the system) minimizes it to a card, your app should throttle back and run in a more conservative manner. What do we mean by that? Here's an example. If your app usually requires near-continuous network access while running, poll the network while your app is maximized and in use...but if your app gets minimized to a card, chill out! Either discontinue or seriously reduce the number of times you poll the network. Doing so will allow the app that is maximized to make the greatest use of the system...and that's the app that the user cares about the most. The one they're using right now.

Let's Review

The Email 1.0 team did a very good job of creating a clean and simple application that accomplished their goals, and supported the user's workflow simply and elegantly. You can do it when you design your app, too...and it all starts with a few great ideas, some paper and pencil, or a whiteboard.

Start prototyping

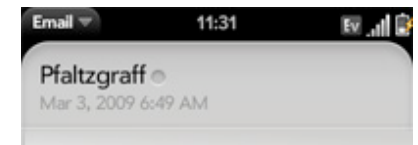
Each scene in a webOS application is very similar to a page in website design, or a view controller in a desktop application. In the previous example, you learned how to structure your application and when you should open a new card to display a scene in vs. when you should display scenes in the current card.

Now you're ready to start designing the scenes you will use in your application. Scenes commonly contain things like a Header, a Scroll View, one or more buttons in the Command Menu area, and other commands in the Application Menu. The webOS platform contains the Mojo Framework, which provides you with many UI controls you can use in each of your scenes. Use the ones you need in order to provide your users with tools they will need to succeed while using your application.

Headers and Commands

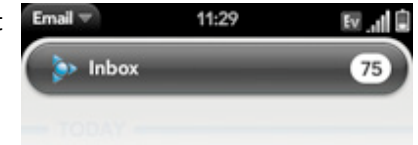
Scrolling Header

This is the area at the top of the scene containing the title, and may include other controls or information (like at the top of an email message). A scrolling header scrolls out of view when the user scrolls the scene. If a header does not need to remain in place, use a scrolling header. Implemented using the **palm-page-header** framework class.



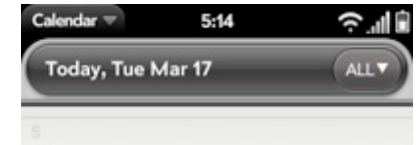
Fixed Header

This is the area at the top of the scene containing the title. A fixed header never moves. When the user scrolls content in the scene, it scrolls under the Fixed Header. Implemented using the **palm-header** framework class.



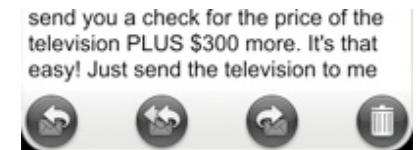
View Menu

A View Menu automatically displays a Fixed Header with a pop-up menu. The items in the pop-up menu in the right-side area control the contents in the scroll view below. Implemented using [Mojo.Menu.viewMenu](#).



Command Buttons

These buttons appear in the Command menu area at the bottom of a scene. Use command buttons for the most important actions performed in a scene. Buttons can display text labels, icons, or both. Maximum five buttons fit comfortably in the Command menu area. Try to use fewer if you can to keep the UI simple and uncluttered. Implemented using [Mojo.Menu.commandMenu](#).



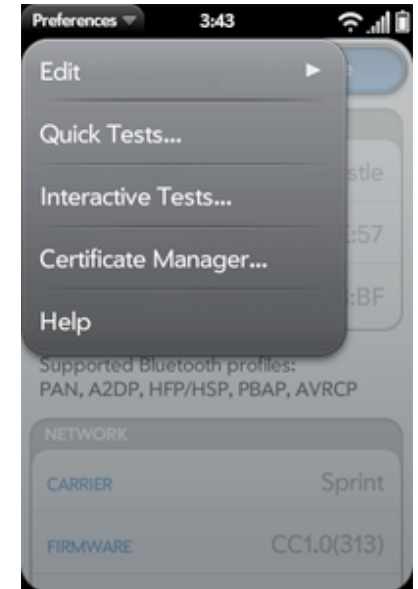
Tips: To create Command Button icons, make a 32 by 64 px (W x H) PNG file (24 bit/pixel RGB with a 1-bit alpha channel). Draw a 24 by 24 px monochrome image in the top 32 px for the unpressed state, and draw 24 by 24 px for the Pressed state in the lower 32 px.

Application Menu Items

You can insert additional commands in the Application menu in each scene. You can also add submenus, if needed. Keep command names short and use title capitalization (like for a book) when naming them. Implemented using [Mojo.Menu.appMenu](#).

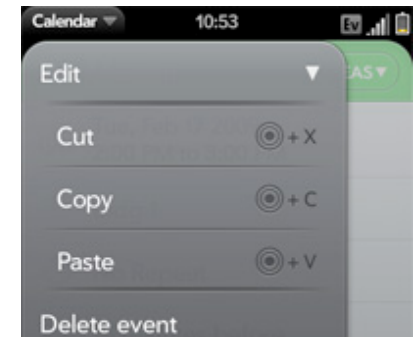
By default, the Application menu includes:

- Edit:
 - The first menu item in all scenes.
 - Contains the Cut, Copy, and Paste submenu commands.
 - Enabled when interacting with a text field.
- Scene-based commands:
 - An application can insert items into the Application menu. Try to add no more than four commands. If adding more than four, the user has to scroll the menu list.
 - Keep command names short and direct, and use title capitalization for the titles.
- Preferences and Accounts:
 - Appears in every scene's Application menu when there are preferences and/or account settings used to connect to one or more services.
 - If there are no preferences or account settings, exclude this menu item from the Application menu in all scenes.
 - If your application has either Preferences or Accounts, include the command in all scenes and name it either "Preferences" or "Accounts." If your application has both Preferences and Accounts, display "Preferences" first, followed "Accounts."
- Help:
 - Opens the Help feature for your application.



Submenu

If a command has choices (e.g., the Edit command), use the Submenu feature. To display the submenu items, tap the submenu name.



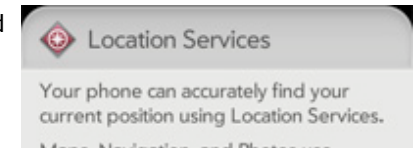
Text and Buttons

Title

This is static text in a scene that serves as a title. Implemented using **palm-body-title** style.

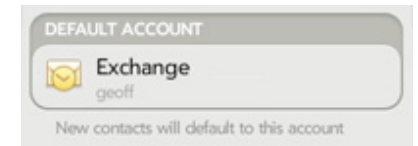
Body Text

This is static text that explains something in a scene, such as the text in the bottom half of the example. Implemented using **palm-body-text** style.



Information Text

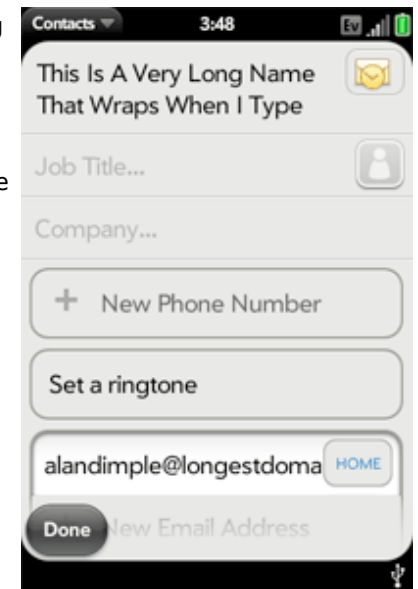
This is static text that provides further information about a control in a scene, such as the text regarding "New contacts ..." in the example. Implemented using **palm-info-text** style.



Text Field

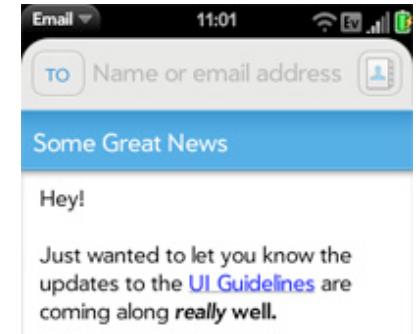
This is an entry field that a user can enter, select, copy, and paste text. There are three variations. Implemented using the [Mojo.Widget.TextField](#).

Set attributes on a text field to set single line versus multi-line, autoTruncation, autoGrow, and more. When an entry field is active, press the Sym key to display a pop-up that displays special characters. Vertically scroll through the list, and tap one of the characters to insert it. The animation on the right shows a field that autoGrows when entering more content and a single-line text field that autoTruncates after committing to a value.



Rich Text Fields

This is an entry field that a user can enter, select, copy, and paste text. In addition, the user can apply bold, italic, underline, and color formatting to the text. Implemented using the [Mojo.Widget.RichTextEdit](#).



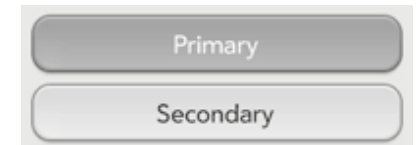
Password Field

This is an entry field that only displays the character you just typed. It displays all previously-typed characters as dots. The last character automatically turns into a dot after a few seconds. Implemented using the [Mojo.Widget.Password](#).



Push Button

Use a push button when the user needs to trigger an action. When pressing a button, it appears "pushed" in and then returns to its unpressed state. Implemented using the [Mojo.Widget.Button](#). There are various types of standard buttons, such as Primary/Secondary buttons, Affirmative/Negative buttons (for constructive/destructive actions), and so on.



Value Selectors

Checkbox

This is a control that turns a value on or off. It appears as a box with two states: unchecked ("off") and checked ("on"). Implemented using the [Mojo.Widget.Checkbox](#).

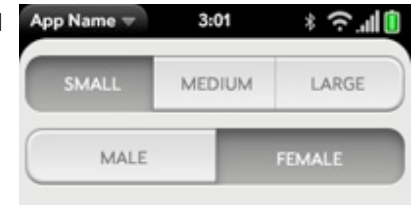
When labeling a checkbox, use words or phrases that clearly indicate what the on and off state means. For example, "Display alerts" makes sense on a checkbox because the "on" state means "display alerts" and "off" means "do not display alerts." Use checkboxes anywhere in a scene. To display several of them, it helps to display one checkbox per line in a **palm-list** control for easy tapping of the checkbox target.



Segmented Buttons

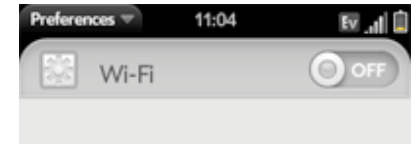
Use a segmented button to offer users a set of options (buttons) where only one may be chosen at a time. Segmented buttons are displayed in a group. Implemented using the [Mojo.Widget.RadioButton](#).

Examples: In the first segmented button group, the pressed button is Small and it appears pressed in. Pressing another button causes Small to reset to an unpressed state and that next pressed button to appear pressed in. In the second segmented button group, the pressed button is Female.



Toggle Button

This is a control that turns something on or off. They look like traditional switches, and have two states: on and off. Implemented using the [Mojo.Widget.ToggleButton](#). When labeling a toggle button, use words or phrases that clearly indicate each state. For example, the label "Wi-Fi" makes sense on a toggle that looks like a switch (you can turn that service on/off).



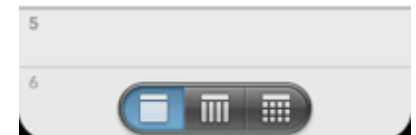
MultiToggle Button

This is a control that toggles between multiple states. Successive presses cycle through the options. The Music application is a good example. Load some music files, press Shuffle All, and then tap the icon in the lower right-side corner. It toggles between three states: Play all songs once, Repeat all, and Repeat 1 song.



Command Button Groups

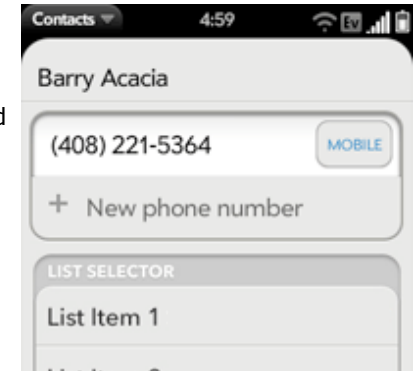
This is a set of two or more command buttons displayed together, and only one can be pressed at a time. Command button groups set the context of a scene or set the value in a controller. Implemented by adding a Menu Group to the Command Menu area, and then adding menu commands or icons to the group. The example shows how the Calendar application uses this group for switching among the Day, Week, and Month View. Implemented using [Mojo.Menu.commandMenu](#) with icons in a Menu Group.



Pop-Up Menu

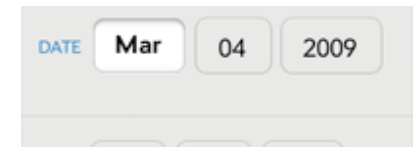
This is a menu that pops up when tapped, and displays a set of available choices where only one can be chosen. Implemented using the [Mojo.Widget.ListSelector](#).

Example: Used in Contacts. Edit a contact and enter a phone number. Tap the inline pop-up menu to declare what kind of phone number this is (e.g., home, work, etc.).



Date Picker

This is a set of pop-up menus that appear next to each other displaying the options for month, day, and year. Tap on each menu to choose a value, or type while the focus is on the control and enter a value by using the keyboard. Implemented using the [Mojo.Widget.DatePicker](#).



Time Picker

This is a set of pop-up menus that appear next to each other displaying the hours and minutes with an AM/PM picker. Tap on each menu to choose a value, or type while the focus is on the control and enter a value by using the keyboard. Implemented using the [Mojo.Widget.TimePicker](#).

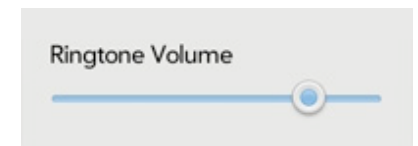


Integer Picker

This is a pop-up menu that contains a set of numeric values. You set the minimum and maximum values. Tap the menu to choose a value, or type while the focus is on the control and enter a value by using the keyboard. Implemented using the [Mojo.Widget.IntegerPicker](#).

Slider

This is a horizontal slider with a control knob that you tap and drag to the desired location. Specify the minimum (left side) and maximum (right side) values. Optionally indicate intermediate choices that trigger additional behavior. Implemented using the [Mojo.Widget.Slider](#).



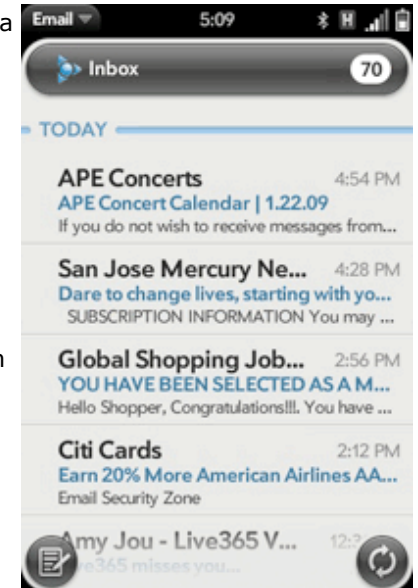
Lists

List

This is a list that displays a set of available choices where only one can be tapped. Tap an item to select it (serving as a single-selection list) or trigger an action (acting as a selector). Implemented using the [Mojo.Widget.List](#), with contents in palm-rows, and by using various options including:

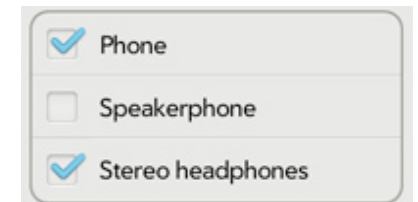
- Add another item to the list
- Alphabetical dividers
- Drag to reorder items
- Static or expandable/collapsible dividers
- Swipe to delete items

A row can have a single touchable item or multiple elements, each of which processes a tap differently. In a simple list, the entire row is touchable and leads to a scene where details are displayed (like in the first scene in Tasks 1.0). In more complex designs, each row can contain several different elements. Example: at a movie website, each row might contain a thumbnail of the movie (touchable, to watch the trailer), followed by two lines of information to the right (Movie name on line one, Rating on line two, not touchable so user can easily flick to scroll) and a button to the right, which the user taps to read details about the movie.



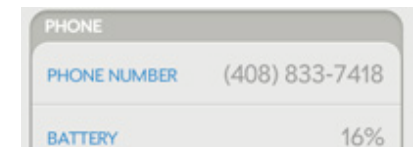
Multiple Selection List

This is a list that displays a set of available choices where more than one can be chosen. Tap the item to select it. Implemented using the [Mojo.Widget.List](#) using palm-list, with a checkbox in each palm-row.



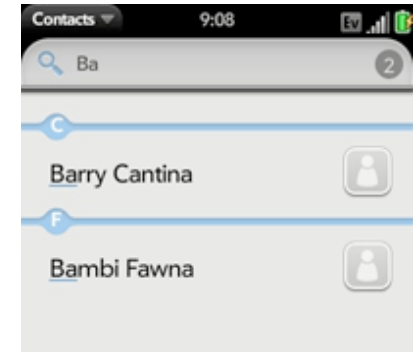
Group Box

This is a list that is enclosed in a border. Most group boxes have titles, but they are optional. Commonly implemented using the [Mojo.Widget.List](#) using palm-group attribute.



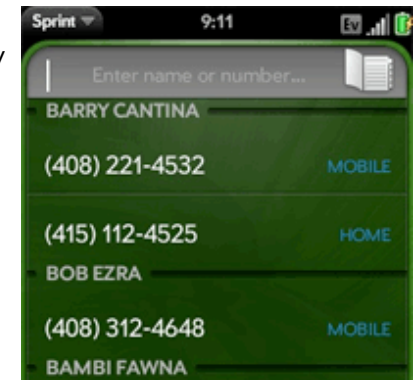
Filter List

This is a control that includes an entry field and a list. Typing in the entry field filters the list to show only the items that match. Use attributes in the FilterList widget to specify how to display the results. Implemented using the [Mojo.Widget.FilterList](#). Example: In Contacts, with a full list of contacts displayed, begin typing on the keyboard. The list is filtered based on the typed entry.



Filter Field

This is a control that accepts text input and passes it to any specified list. Implemented using the [Mojo.Widget.FilterField](#) using `palm-group` attribute. Use the Filter Field to process the text the user enters, and display the results however you want.

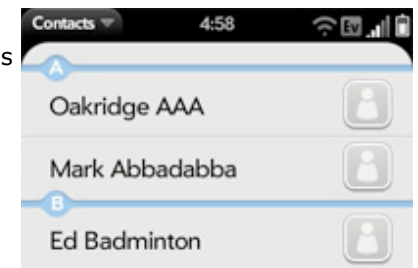


Visual Organizers

Advanced applications may contain lots of different UI controls, which require organization. The webOS development environment provides controls that you can use to organize controls within a scene.

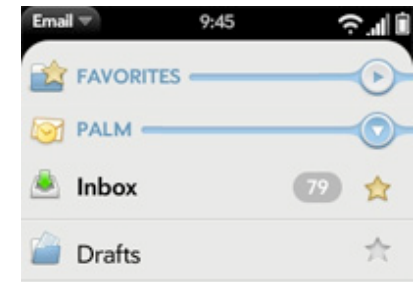
Dividers

A divider is composed of a simple line or a label with a horizontal line to its right. There is a variety of display styles. They are used as a header for a group of information. Example: An alphabetical list of contacts, where each divider has a letter to organize contacts by their last names. Implemented using the [Mojo.Widget.List](#) using the divider attributes.



Collapsible Dividers

A collapsible divider has a label followed by a horizontal line and then a button. Tap the button to open and display items under that divider, and then tap again to close it. Implemented using the [Mojo.Widget.List](#) using the divider attributes. Example: Open the Email application. That app uses collapsible dividers in its first scene to allow a user to hide/show all of the mail folders under each account.

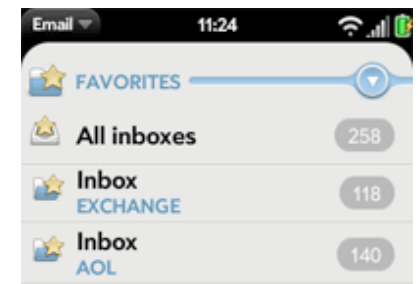


Drawers

This is an item that shows and hides the items contained within it. Implemented using the [Mojo.Widget.Drawer](#).

Examples:

- The Email application allows users to hide and show the folders (displayed in a List) under each Email Account divider.
- The Application Menu contains an Edit command that displays its submenu items for Cut, Copy, and Paste using a Drawer
- The Contacts application allows users to "link" multiple contact profiles together to collect all of the information you have about a person from multiple sources into a single contact record.



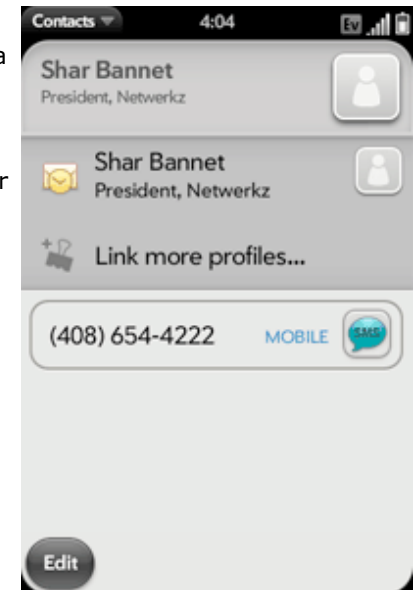
Connecting to Data and Services

This allows your application to make intelligent use of data and services from other applications on the platform.

People Picker

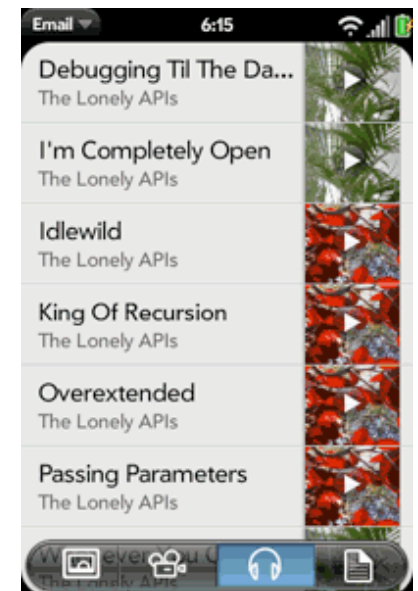
This allows your application to display a list of records from the Contacts application, allowing the user to add/associate/use their data in your application. It auto-filters a typed list, or you can scroll through the list to select a person. Implemented using the [Mojo.Widget.PeoplePicker](#).

Example: In the Contacts Application, launch Contacts and tap a person for whom you have information coming in from two different accounts (Corporate Server and Facebook®). The example on the right shows a person named Shar Bannett. Tap the picture icon, and then tap "Link more profiles" (the list you see is a People Picker). Type letters using the keyboard to filter down the list to match what you type. Choose another profile to attach to the current contact (Shar Bannett).



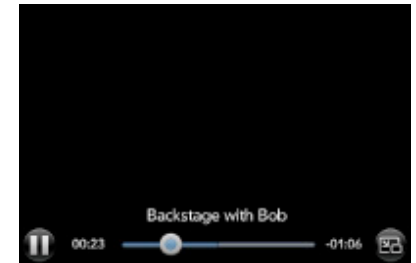
File Picker

The file picker is a full-screen compound element that allows the user to attach or choose one or more files for use in your application. The Email application uses this to allow users to browse files for attaching to an email message. By default, it shows every file type and provides controls for selecting them (e.g., pictures, movies, songs, and files). You can specify criteria if you want the file picker to only include specific kinds of files for use in your application. Implemented using the [Pickers](#).



Media Player

The Media Player is a full-screen widget that allows the user to play media files by using standard controls within the context of another application. For example, opening a music attachment in an email or a music file in the browser. This widget opens a new scene in the current window. Implemented using the [Video Playback](#).



WebView

This is a control that allows you to embed an object that displays Web content in a scene. It can take over the entire scene or be rendered in a frame of a fixed size. It can display content from an external URL or URL-addressible local content. Implemented using the [Mojo.Widget.WebView](#).

ImageView

This is a control that allows you to display one or more images. Provides support for panning, zooming, and navigating from one image to the next. Implemented using the [Mojo.Widget.ImageView](#).

The Application Menu

The Application Menu allows you to create special menus that appear in each scene. This is a very powerful concept. The Application menu is a dynamic menu that contains the following menu items by default:

- **Edit**, which is always the first menu command
- **Preferences & Accounts**, if applicable for your application
- **Help**

Adding Commands

You can insert each scene's special commands between Edit and Preferences & Accounts on a scene-by-scene basis. You can add up to 4 additional commands to the Application menu comfortably. If you add more, the user will have to scroll to find the rest of the commands. We recommend against this. Also be aware that you can add a submenu item in the Application menu. Each submenu can contain a set of commands. The webOS platform contains a submenu: the Edit menu, which contains the Cut, Copy and Paste commands.

Command names should be written in title case and include an ellipses (...) if the user must take further action after selecting that menu item to complete the named task. For example, a menu item named "Add Bookmark..." which opens a dialog that must be completed before the bookmark is added should have ellipses. A menu item named "Bookmarks" which opens a new scene containing bookmarks should not have ellipses, since once that new scene is opened, the named action is complete.

The menu for a given screen may change depending on the state of the application and view.

- Items may change their visual state to represent enabled and disabled states.
- Items may be suppressed entirely when not relevant.

As a general guideline, applications should show a menu action in the disabled state when:

- That action is commonly available in the circumstances in which it appears currently. For example, most applications can be deleted in the launcher. Applications that cannot be deleted should show the delete menu action in a disabled state, instead of suppressing the action entirely.
- It's important to communicate the fact that that action would be available, but is not available in the current circumstances.

Applications should suppress a menu action entirely when:

- Even though that action may be available in other similar circumstances, it's not expected that the action will always be available. For example, a "share" menu action may be commonly available in detail views, but the fact that it appears in the Photos application does not set a precedent for it to appear in the Videos application.
- Users will not expect the action to be available.

Other reasons why menus for a given screen may change include:

- Items may be replaced by different items. For example, the speakerphone icon might be replaced by a Bluetooth submenu when the device is connected to a Bluetooth device.
- Item's internal state may change. For example, I might have an image that changes to reflect the ambient light, or how much the accelerometer is bouncing.

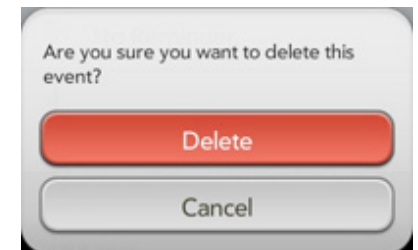
The Edit menu item is always present, to preserve menu layout for faster scanning. However it may be disabled (grayed out) depending on the state of the scene. Submenu commands (Cut, Copy, and Paste) may be enabled, depending on the state of the page and the clipboard. If all submenu commands are disabled, then the Edit menu is disabled (and does not open).

- **Paste** is only enabled if the clipboard contains data that the scene's current focus can accept.
 - For webOS platform release 1.0, this is limited to text (on the clipboard) and focused text edit fields (in the scene).
- **Copy** is only enabled if text is selected, or if an app decides to expose it's entire scene as a text equivalent.
 - For example, a contact detail card may push a text version of its content to the clipboard in response to a Copy command.
- **Cut** is only enabled if text is selected in an editable field.

Providing Feedback

Dialog Boxes

This displays a message in a dialog panel when your application needs the user to make a decision so it can continue. A Dialog Panel rises from the bottom of the current scene, and is attached to it. It can be displayed when your application is in the foreground, or when minimized to a card.



Progress

Display a Progress or an Activity Indicator if a process is going to take two seconds or more to complete.

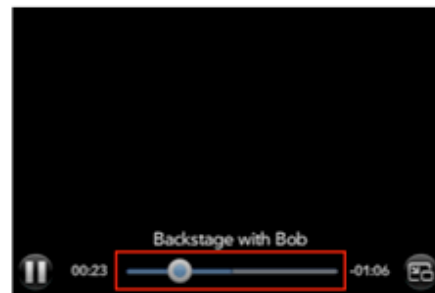
Progress Indicators

Display a progress indicator when you know the timing length of the task and the amount that has completed. It displays the amount of a task that is currently complete, given the task's entire length. There are several kinds of progress indicators, which are:

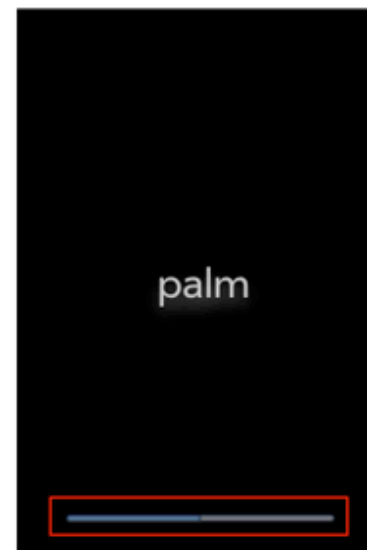
- **Progress Pill** — Use this when you need to show download progress, loading from a database or progress during a long operation, and you know approximately how long the task will take. The horizontal bar in the pill updates as the task progresses. Tap the X to cancel the task. Implemented using the [Mojo.Widget.ProgressPill](#).
- **Progress Slider** — Commonly used in audio/video applications where you know how long the content is, want the user to know where they are in the playback process, and allow them to control it. The tracking slider moves as the content plays. Drag it to move the "playback head" or tap anywhere along the slider to reposition it there directly. Implemented using the [Mojo.Widget.ProgressSlider](#).
- **Progress Bar** — Display this control when you need to show the progress of a task (e.g., launching, initializing, etc.) and you know approximately how long it will take. The progress bar simply shows progress. It does not allow the user to cancel the task. Implemented using the [Mojo.Widget.ProgressBar](#).
- **Inline Progress Bar** — Commonly used in lists that display the names of song files. Allows the user to tap the list item and hear a preview of the item.



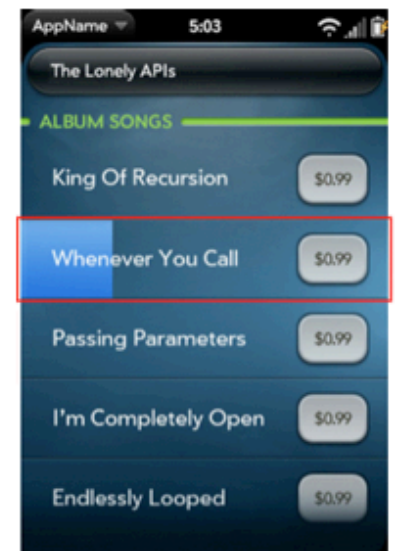
Progress Pill



Progress Slider



Progress Bar

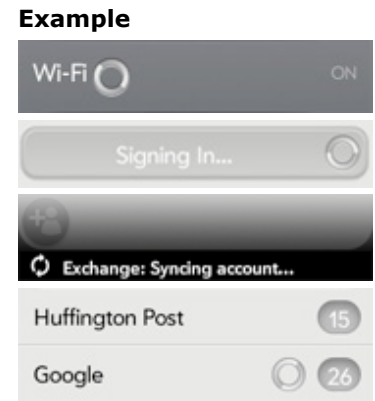


Inline Progress Bar

Activity Indicators

Display an activity indicator when you cannot make a reasonable guess as to how long a task might take. An activity indicator spins indefinitely, letting the user know that the application is making progress, but gives no indication as to how much progress has been made or how much more work remains to be done. They come in large and small sizes, and can be displayed in many situations. They are implemented using the [Mojo.Widget.Spinner](#).

Where	When
In a menu item	When the user presses a command that must be processed, like when turning Wi-Fi on.
In a button	When the user presses the button to log in to an email account for the first time.
In the notification bar	When the system gathers data from the email account for the first time.
In a list	When it may take awhile before the displaying the tapped item. Display a small spinner in the tapped row. If tapping anything else on the screen could cause performance or the response time to worsen, display an intermediate "progress" scene that displays the progress with regard to the tapped item instead. Avoid simply displaying a large spinner over the list.



Errors

Report an error when the problem impacts the user experience and the application cannot resolve the issue quickly on its own. Consider the recommendations in the following table.

Topic	Recommendation
Banner notification	Use this location when the user only needs to know about the problem, but does not need to take action.
Dialog panel	Use this location when the application cannot continue without a decision by the user. When writing the message and naming the buttons, remember to phrase the message as concisely as possible so the user understands the issue and what actions they can take. Also, avoid the word "please," negative phrases (e.g., "invalid account" or "wrong password"), and technical phrases (e.g., "server error" or "timed out"). A good example is "Try again." Provide buttons that make it simple to take corrective actions. Use verbs or phrases for the button titles that correspond directly to the language used in the message. If any of the actions are destructive, use the "destructive" button style and provide a CANCEL button.
Inline message	Use this location to display an error alert icon and a message in the scene, under the tapped item that generated the problem. If multiple items contributed to the problem (e.g., incorrect username and password), display a single message summarizing how the user can correct the problem.

Notifications

The webOS platform allows applications to display messages at any time and, most important, display them in an area that doesn't interrupt or intrude upon the work that they're currently doing. Your app can display banner notifications and dashboard summaries in these kind of situations:

- **When your app is running (but is minimized) or is a "Dashboard App"**

- Display Banner Notifications to let users know when your app does something you think they should know about. The notification will appear at the bottom of the screen. It will not overlay the application the user is currently using. It will minimize to a dashboard summary icon a few seconds later.

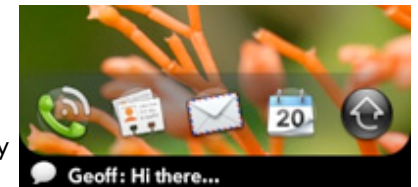
- **When your app is maximized (running and is frontmost):**

- Use Banner Notifications to provide users with status when they use your application to begin a lengthy process that can continue in the background, while the user does other things in your application.
 - Example: Imagine you're using the Email app and it's currently set to display email from one of your personal accounts (like AOL, Google, etc). Then you add your work email account, which connects to your company's corporate "Exchange" mail server. It can take awhile to connect to that account for the first time and copy the important information to your phone. Email could display connection and download status as banner notifications to show the user how the app is progressing...and would still allow the user to read email from their personal account.

Banner Notification

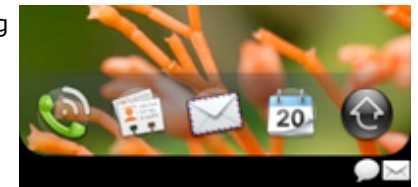
Banner Notifications appear under the current application and usually consists of a dashboard summary icon followed by a message. The Messaging application does this when running in the background and someone sends you a text message. It displays a dashboard summary icon on the left, followed by the sender's ID and a portion of the text message.

Displaying Banner Notifications is optional. Some applications (like Email) have a preference the user can set to display banner notifications, and it is off by default. Email simply displays the dashboard summary icon on the right immediately instead.



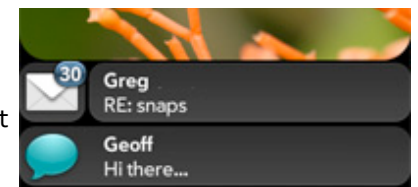
Dashboard Summary

If your application displays a Banner Notification, it disappears after a few seconds and slides to the right, representing the notification using a dashboard summary icon. The example shows two dashboard summary icons: one from Messaging and another from Email.



Tap a dashboard summary icon and the Dashboard View appears, displaying detailed information about each dashboard item. When displayed, the user can flick each dashboard item to the right to dismiss it or tap any part to open it. Your app is responsible for the icons and text displayed in Dashboard Summary items.

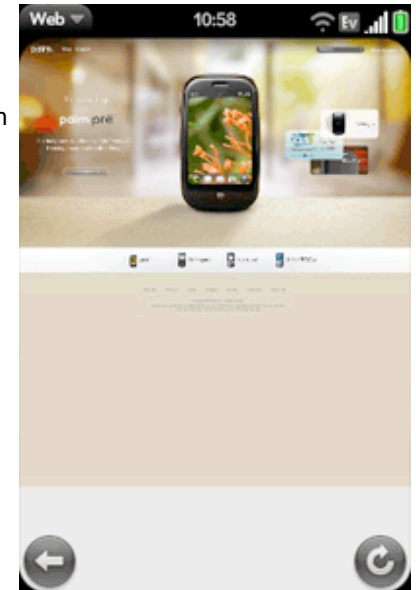
The example on the right shows two Dashboard Summary items: one that represents a single item (incoming text message from Geoff) and another that represents many items (30 emails). If your application processed a single event that should be reported, display it as a single tapable item in the dashboard. If your application has accumulated multiple items while running in the background, display them using two tap targets, similar to the way the Email app displayed its Dashboard Summary Item. Tap the part on the right to open the most current item or tap the part on the left to see the list of emails.



Pop-Up Notification

If your application really needs to get the user's attention while it is in the background, use a Pop-up Notification message. It slides up from the bottom of the screen and displays the options available to the user.

Example: Using one application (like Web) and a Calendar Alarm goes off. The Calendar app displays a reminder, which the user must interact with before it is dismissed. The Palm webOS™ platform displays Pop-up Notification, too. You can see one when plugging a USB cable into your phone and computer. The Pop-up Notification asks what you want to do, mount the phone as a USB drive or simply charge the device. You are responsible for designing and implementing the UI that appears in Pop-up Notifications. When you design them, make them as small as possible and never exceed more than half of the screen's height.



Dashboard Applications

"Dashboard applications" are apps that provide a service of some kind (e.g., stock ticker, current weather, surf report) and don't need a very elaborate UI in order to do so. Applications like this typically have a single card that users use to configure the service and Dashboard Notifications, Dashboard Summary Icons and Popup Notifications to display information in when its time.

Dashboard apps must have an application icon and some kind of UI manifestation so users can:

- configure the application, in order to determine what the app should display and when
- be notified when important information arrives or events occur
 - notifications, dashboard summary icons and dashboard items, in which to display important information
 - popup notifications, in which to display urgent information
- quit the application

Layout and Customization

The visual elements of the webOS platform were designed to make it very easy to make a great looking application with any extra effort. By default, your application will have a light gray background and will display text and controls that contrast properly against that background. Furthermore, they were created at sizes that ensure their touchability and visibility.

When the webOS controls and fonts were chosen, the designers kept the following things in mind:

- **Control sizes and layouts were optimized for "touchability"**
 - Ensures that controls are the right size, and that hit targets are set so each item can be tapped successfully
- **Controls and text were designed to ensure an appropriate amount of "whitespace" around each control**
 - Ensures that controls are spaced properly, for optimal readability.
- **Colors and Font sizes were chosen and are used consistently to communicate specific messages, such as:**
 - enabled vs disabled states, which indicate which items can be tapped (enabled) vs. those that cannot (disabled)
 - which buttons are of primary vs. secondary importance, or constructive vs. destructive
 - Dark gray buttons to indicate the primary action, and lighter gray buttons are used for other actions
 - Red button are used to indicate actions that are destructive. Green buttons indicate that an action is constructive (or positive).

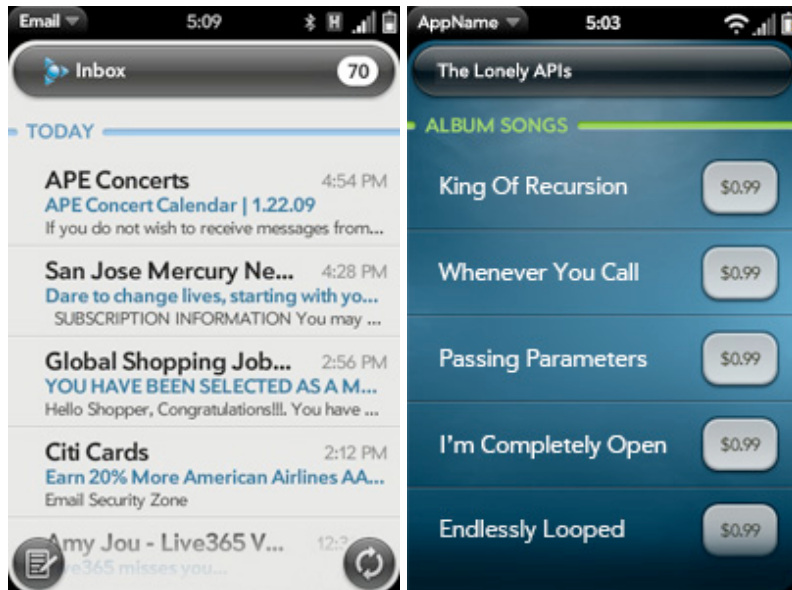
If your application uses a light-colored background with dark text/controls, the default controls and text colors should work very well for you. If your application uses a dark-colored background with light text/controls, use the **palm-dark** styled controls and text. Some of the applications Palm ships on the phone use the palm-dark controls (Music, Videos).

If you want to customize the look of your application, here are some things you can do:

- **Change the backdrop**
 - Sometimes a simple change like this can make a world of difference for your application. Add your new backdrop and use either the default or palm-dark style controls with it. Chances are, the text and button styles in one of those styles will work very well against your backdrop.
- **Fonts**
 - In order to provide a consistent User Experience and ensure readability, Palm uses certain font sizes for certain kinds of text. It's okay to adjust the font colors to work with your look and feel, but do not change the font sizes. If you customize your font colors:
 - Make sure that colors you choose for contrast properly against your background.
 - Be sure that you the colors you choose properly convey the right information
 - Enabled items use colors that contrast highly against the background (example: white text against a black background)
 - Disabled items use colors that have lower contrast against their background (gray text against a black background)

Default Style

Dark Style

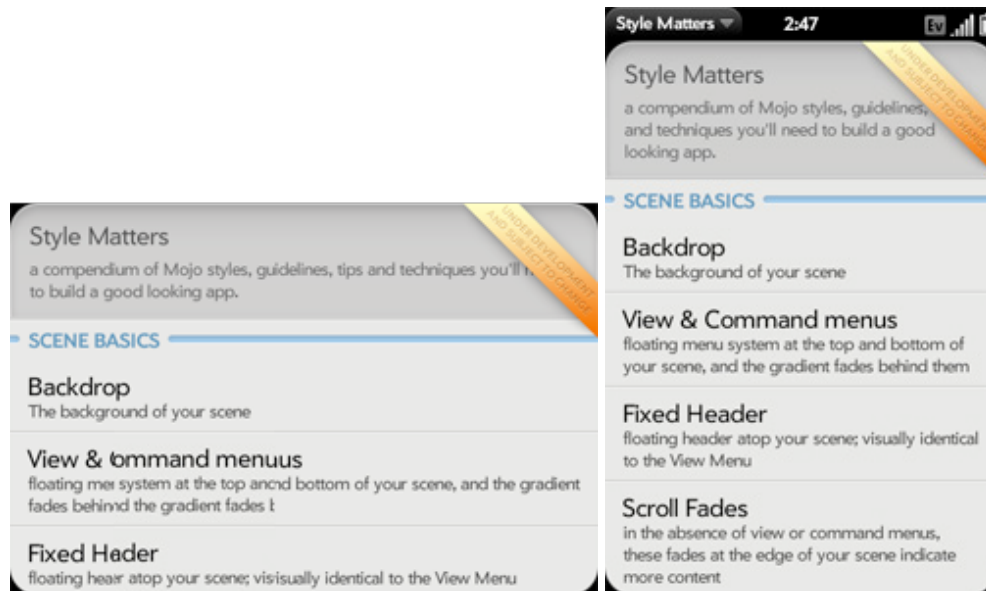


Layout

Because the webOS devices reports their orientation, your application can take advantage of this information and display the contents of its scenes differently when the user rotates the device from portrait orientation to landscape orientation and back. If you want your application to respond to rotation changes, set the size of the elements in your scenes so they "do the right thing" when the device is rotated.

Landscape Orientation

Portrait Orientation



The "Style Matters" application responds to rotation, as you can see in the example above. This application did this by setting the size of their scene elements "flexibly" (rather than by using fixed heights or widths). For instance, the width of the header and list is set to occupy 100% of the width of the screen, and the height of the header is not set at all. Setting heights and widths flexibly makes it possible for the scene to display content intelligently, making the most of whatever screen height or width is available at any time.

Usability Testing

The only way to really find out if *regular* users can use your product is to let them try it. Test your product early, and test it often. Plan time to incorporate the feedback that you get after each round of testing. When testing, keep the following things in mind:

- **Find "regular people" to try your application**
 - While it's fun and rewarding to have your friends and colleagues try your app, the only real way to get real unbiased feedback is to get "other people" to try it. There are many ways to find them.
 - Write a short ad and post it on free online community services, like craigslist. Recruit people who either have a webOS device or a competing maker's device and would like to try an app like yours. Offer some form of payment (money, gift card, or something tasty...like a half-caf double mocha latte).
 - People use mobile devices on-the-go. Have them try your app wherever they'd normally use it (or in a place like that). Here are some suggestions: At the park, on a couch, standing in a line, riding in a car/bus/train...or at a coffee shop, sipping the aforementioned half-caf double mocha latte.
- **Ask users to perform meaningful tasks**
 - Flash back to your mission statement. You know what you designed your app to do. Ask users to do those things. See if they can, quickly and easily.
 - Let users poke around afterwards, to see what they'd do with your app on their own.
- **Watch and listen closely**

- Watch users while they work.
- Let users struggle a little while they work. See if they can figure out what to do without your help.
 - If they struggle for "too long" or are getting frustrated, ask them what they're trying to do.
 - Use that opportunity to find out what was difficult for them.
- **Look for trends**
 - Record the problems that you discover in each test
 - If many users experience the same problems, there's probably a problem with the UI design or the implementation.
 - If only 1 or 2 people have problems, try to find out why. Ask them questions while meeting with them.
- **Create solutions**
 - Usability Testing helps you find the "biggest obstacles" in your UI. Brainstorming afterwards helps you come up with great ways to fix them, using standard UI controls and interactions wherever possible.
 - Feel free to brainstorm with users after a usability study, but remember that you don't have to do everything each one of them says. Take their suggestions as input, consider it and go with the ideas that make sense, and are consistent with the way other apps on the webOS platform work.
- **Fix the problems and test the app again.**

Prepare for Delivery

Once your application has been developed, user tested and unit-tested for quality, you're ready to go. Here are the last few items you need to attend to before uploading your application to the Palm App Catalog.

- **Create your application icon**
 - 64 by 64 pixel PNG file, with a 56 x 56 pixel image centered within the frame
 - 48 by 48 pixel PNG file, with a 42 x 42 pixel image centered within the frame
 - 24 bit/pixel RGB with a 1-bit alpha channel
 - Do not render a "glossy shine" like applications on other mobile OS platforms. Instead, place your application image on a rectangular base (like Music or Videos apps) or use a photo-realistic image with a transparent circle behind it (like Web, Camera or Photos).
- **Make sure it's ready**
 - Check your app using the [Application Checklist](#)
 - If you can check all of the items on the list, you're ready to submit it!
- **Submit it**
 - Upload your app to the App Catalog
 - Write a description
 - Tag it with keyword descriptors, so users can find it when they search
 - Add Screenshots
 - press Orange+Sym+P. The screen capture will be in the Screen Captures folder when you mount your device to your computer as a USB drive.

Every webOS device ships with the App Catalog application, which users will use to find your app. And we hope they do...so get to work and create a truly amazingly great application! The world is waiting.