Graphical Browser Application Style Guide

Openwave Mobile Browser, WAP Edition 5.0

Openwave Systems Inc.
1400 Seaport Boulevard
Redwood City, CA 94063 U.S.A.
http://www.openwave.com

Part Number MBWS-50-002
Openwave Mobile Browser, WAP Edition 5.0, August 2001

Contents

1 Overview

This document gives developers who are writing content and applications a comprehensive guideline for developing highly usable applications that run on Openwave Mobile Browser, WAP Edition, 5.0 graphical browsers. Applications that are written to take advantage of the graphical Mobile Browser elements will not run on other browsers in the market; therefore it is necessary to develop or maintain applications for the Openwave Mobile Browser, 4.x and other nongraphical browsers. This document considers various situations and possibilities, offers the most usable solutions, and provides sample code.

Organization

This document begins by outlining some usability philosophies and then gives detailed guidelines on navigation, selection lists, text display, forms, alerts, and many more topics. Recommendations that are specific to the graphical Mobile Browser are preceded by "(GUI)."

This guide also has three appendixes:

- Appendix A, "Identifying the Browser," describes how to distinguish between various browsers.
- Appendix B, "Summary of Graphical Mobile Browser Elements and Attributes," is a reference table of the new graphical Mobile Browser attributes and elements, along with tips on when and how to use them.
- Appendix C, "Sample Application," provides a sample application, a design review and summary, and a technical summary. The specific content of the sample application may not apply to all environments. The sample does not specify how applications should work; it simply illustrates good design practices.

The information in this document is derived from usability tests, knowledge of the capability of the WML, and testing applications on a variety of phone models and SDKs.

Differences Between Text-Based and Graphical Browsers

There are two versions of the Openwave Mobile Browser, 5.0: a graphical browser and a text-based browser. Both browsers are WAP June 2000 (WAP 1.2.1) compliant; however, the graphical browser has additional features that you can use to improve the look and feel of your application. The graphical Mobile Browser offers improved support for the following user interface features:

- Titles
- Forms
- Tables
- Multiple-selection lists

The following features have been added to the browser to enhance the user experience:

- Buttons
- Text boxes
- Radio buttons
- Pop-up menus
- Pop-up menu from the secondary softkey
- Horizontal rules

Many Openwave licensees are working on developing handsets with the GUI browser. However, many handsets in the market still have the text-based browser. Therefore, to ensure usable applications, you must still write and maintain applications for those browsers. An application that is written for the 4.x browser will still run on both versions of the 5.0 browser. However, there are some additions and changes that you can make to enhance usability for the 5.0 graphical browser. This document focuses on designing and writing applications specifically for the 5.0 graphical browser and provides information about when and how to implement the new features. For information about the 5.0 text-based browser, refer to the Openwave *WML Application Style Guide*, available at `http://developer.openwave.com/support/techlib.html#styleguides`

Table 1-1. New graphical Mobile Browser elements

Element/Attribute	How to use	What to avoid
Card titles	Limit titles to 15 characters.	Do not make titles longer than 15 characters because they may be truncated on some devices, causing the title to be confusing or ineffective.
Tables	Use tables to present columnar data and to align columns of text.	Do not use so many columns that the data wraps past one line.
Buttons	Place buttons at the end of a card to complete an action and take the user to another URL. Use buttons in the beginning or middle of a card only to allow users to fill in or select data to be used in the originating card.	Do not use a button in the middle of the card to submit the contents of a form and access another URL. Avoid icons on buttons except in certain cases, such as using a calendar icon to access a calendar.
Text boxes	Use growing text boxes instead of fixed-size text boxes if the information is likely to exceed one line. Use growing text boxes if the size or amount of data to be entered is not fixed. Use fixed-size text boxes if the `maxlength` attribute restricts the length of the field to 12 characters or fewer.	Do not start a card with an input field followed by a selection list. Do not hide entered text by limiting the size of the text box. Do not define the size attribute greater than 24.
Radio buttons	Use in forms to replace selection list items. Limit the number of radio buttons to 4 or fewer.	Do not define more radio buttons than will fit on one display (approximately 4 radio buttons).
Multiple-selection lists	Use to allow users to select multiple items. Use to turn a setting on or off in a form	Try not to define more than 9 items in a multiple-selection list.
Pop-up menus	Use in forms instead of a selection list. When it makes sense, set the default selection to the one most likely to be chosen.	Do not define more than 9 items in a pop-up list unless it is a list of known items, such as the 12 months of the year.
Horizontal rules	Use to add a visual break between content regions.	Do not define the thickness of the rule to be greater than 1 or 2 pixels.

Testing on SDKs

To help with developing applications for the graphical Mobile Browser, you should use the Openwave SDK, WAP Edition, 5.0. You can download this SDK from `http://developer.openwave.com`.

In addition to testing your applications on the SDKs, you should test them on the browser handsets to make sure that they look right and work correctly. For a list of available devices with the graphical Mobile Browser, see the Openwave Mobile Phone Reference at `http://developer.openwave.com/resources/phones.html`.

2 Usability Design Philosophy

When building an application for a WAP browser phone in the Global System for Mobile Communications (GSM) markets, you should keep in mind the following guidelines. The user's experience with an application determines whether or how often the user revisits the application.

- **Usability is critical.**

  Device constraints limit both navigation and the amount of content that a handheld device can display. Also, users may find it difficult to enter data. Be careful to make your application usable in this constrained environment.

- **Minimize or avoid text entry.**

  Entering text on the phone keypad is tedious. Try to use alternative methods, such as storing previously entered text or data, personalization, and selection lists.

- **Most devices are phones first.**

  Most devices on the market today have added the browser as an afterthought. Neither the hardware nor the user interface reflects up-front planning for the browser. Some features may be more difficult to use on one phone than on another.

- **Mobile handsets are used for information retrieval, not for browsing.**

  Users of WAP browsers do not browse the web as they do with a PC, because the amount and nature of content is scaled down for the handheld device. Phone applications are commonly used for quick information retrieval, not for general browsing and research. Phone users tend to be less technical and in more of a hurry to get to the data they want.

- **A phone is not a PC.**

  The phone has features that a PC does not, such as the ability to integrate voice, data, and alert features. It is important to design applications that address user interaction models, screen size, and screen context. When designing the user interface, keep in mind that there is greater variability among phones than among PCs.

- **Time costs money.**

  Most circuit switch networks charge the user by the minute for browsing on the phone. Assume that users will avoid expensive browsing sessions.

- **Most users avoid complex applications.**

  Applications must be well-organized with a shallow menu structure so that the user can get to the value quickly.

Creating Usable Applications

When developing applications, consider the following important factors:

- Who the user is
- What problems the user is trying to solve
- How to solve those problems most efficiently

This section discusses some key principles for creating usable applications.

- **Specialize your application for the different browsers.**

  To create a usable application, determine the type of browser that is accessing your site. With that information, develop customized versions that increase usability by taking advantage of unique browser features.

- **Know your customer.**

  Users turn to an application to solve a problem and, in some environments, to communicate or to be entertained. For instance, the user's goal might be to purchase an item or to upload and download information while in the field. Build the application to help the user accomplish that goal. If the user's goal is to find a stock quote, display the quote right away. Use the quote display screen as the entry point to any other information the user might want.

- **Get to the value quickly.**

  Deeply embedded information can cause the user to forget the goal, become frustrated, and avoid the application in the future. Provide commonly used options quickly rather than requiring users to navigate deep menus.

- **Limit the application to necessary functionality.**

  Remember that the browser does not have the display and navigation capabilities of a PC. In addition, while browsing on the handheld device, the user wants to find or submit information in the shortest possible time. Scale down the application to meet only the goals of the user, and do not include extras. Provide access to the most commonly used features through menu choices, links, and options.

- **Make the application easy to navigate.**

  Minimize the number of steps it takes to access information. Eliminate or combine cards if this can be done without losing important information, choices, or content. Whenever possible, create multiple paths to access information.For example, if the application provides weather content, allow the user to search by postal code or zip code as well as by city or state. In this way, the user has the choice of entering a short code rather than long strings, which are hard to type on the phone.

- **Make the application consistent.**

  Consistent applications are intuitive for the user. Make the text descriptive and easy to follow. Labels should explain the actions they cause. Order lists logically, so that items and links are easy to find. Although images and icons help make pertinent information stand out, be careful not to overuse them.

- **Avoid text entry.**

  Avoid queries that force the user to enter alphanumeric text. Use selection lists or partial text searches to avoid or minimize text entry.

- **Personalize the service according to the user.**

  Allow an application to retain user information to autofill personal fields. For example, store the login, password, billing address, or other information in a cookie or on the server where the application resides. The user can be determined by the subscriber ID.

- **Anticipate situations in which users are likely to make errors.**

  Make sure that the application does not allow the user to continue a task unless certain requirements have been met. For example, if the user is entering the date, the application should check that the date is 8 digits: 2 digits for the day, 2 for the month, and 4 for the year.

- **Localize the text according to the environment.**

  Make sure that the application is localized to the area where it will be used. For example, in the United States, the term *zip code* is used, and in the UK and Australia, *post* or *postal code* is used. Sometimes the wording can make or break the usability of an application.

Testing the Design

Use display templates to help design the content on each card and the navigation flow for the application. Once the application is designed, use display templates and navigation flow diagrams to show how it will behave on each type of browser. The templates can help provide information on which widget to use, selection lists versus links, text lengths, and wrapping versus scrolling text. For this document, templates restricting the lengths of the text were used to demonstrate menus, multiple-selection lists, nonwrapping text (Times Square text scrolling), the viewing of text and links, and entries. In designing your application, examine every layout and display for consistency. Unnecessary differences, such as the use of scrolling text in one screen and wrapping text in another, can confuse the user.

The sample application diagrams in Appendix C, “Sample Application,” demonstrate navigation only, not other behavior. In contrast, the templates used throughout the rest of the document demonstrate other desired application behaviors, such as wrapping text, softkey labels, and so on.

Graphical Mobile Browser Template

To help you determine the look and feel of your design, see the following Word template:

```
http://demo.openwave.com/styleguide/v5gui
```

This template document provides layouts for the different content displays.

The Openwave Mobile Browser has two programmable softkeys at the bottom—in this example, OK and Done. For these examples, the `<do type="accept">` label appears on the left softkey and the `<do type="options">` label on the right softkey. Each phone has a fixed key for Back or Back/Clear navigation, not shown.

General Graphical Mobile Browser Phone Properties

Although WAP-compatible phones on the market have a variety of screen sizes, number of keys, and key functionality, design your application to display well on a small screen. Applications designed for small screens look good on larger screens, but not necessarily the other way around. In some environments or for some applications it may be helpful to consider classes of devices—for example, handsets of 4 lines versus PDA pen input and selection.

- **Up to 4 lines of text or selection items may be visible.**

  Some phones have as few as 2 lines; some have as many as 8 lines.

- **Approximately 15 characters can be displayed on one line.**

  This number may vary, because many phones support variable-width fonts.

- **All phones have menu up/down navigation.**

- **Right/left navigation may be available.**

- **The browser home deck is available from every card.**

- **Some phones support smart-entry methods, such as T9 or smart mode.**

- **All phones can display graphics or images.**

- **Not all phones have a Send/Talk key.**

- **All phones support uppercase and lowercase fonts.**

- **All phones support links, but phones may display them differently.**

- **Most phones do not have separate Back and Clear buttons.**

  When queries for entries are presented, the user may need to delete all entered characters in order to return to a previous screen.

Browser Properties

The following table shows the properties of the graphical Mobile Browser that you can use to enhance usability. To test applications that implement these features, use an Openwave Mobile Browser 5.0 graphical browser device.

Table 2-1 summarizes the properties of graphical browser devices. (In the table, *device* means a WAP browser phone or PDA.)

NOTE There is some variability among Openwave browser devices; however, this should not affect the user experience.

Table 2-1. Summary of browser device properties

Property	Graphical Mobile Browser device
Number of characters per line	15 characters (mostly variable-width fonts)
Lines of display	3 to 8 lines
Image/graphic support	All devices support images; some devices may support color
Link support	Links are displayed underlined; handsets with color may also display links in color
Scroll key	Scroll keys for up and down; some support left and right scrolling
<do> labels	Label is associated with softkeys on the phone
Menu navigation	Displayed as selectable numbered choice items
Pop-up softkey menu navigation	Displayed as selectable numbered choice items
Back key for navigation	Dedicated Back key available except when entry queries are active
Clear/Back key in entry queries	Shared; user can deactivate an input field to access the Back function
Nonactivated entry field, pop-up menu, radio button, anchor, and multiple-selection list	While the element is selected but not activated, only one softkey is programmable
Activated entry field and pop-up menu	While the element is activated, no softkeys are programmable
Alerts	Both the WAP Push and Openwave alert methods are supported by the device, but the availability to the developer depends on the carrier's implementation

- **A label for the highest priority action is viewable and accessible from all cards.**

  The `<do type="accept">` label is displayed on the screen and accessed by pressing the primary softkey (the right softkey in the template) when no other elements are selected.

- **A label for the secondary actions is accessible from a pop-up menu on the secondary softkey.**

  The `<do type="options">` label is mapped to the secondary softkey (the left softkey in the template) and may require one or more keystrokes to select, depending on the number of `<do type="options">` defined on the card.

- **Each phone has a fixed key mapped to backward navigation.**

  Back functionality `<do type="prev">` is always available from a fixed key on the keypad. The default is the last card in the history if no other location is specified. This key may be shared with a Clear key in entry queries.

- **Bookmarking of sites is supported.**

  Users can bookmark a card from the browser menu unless the application specifies otherwise. Bookmarks can be stored either on the Openwave Mobile Access Gateway or on the device.

- **Most phones allow a press-hold to access the first nine bookmarks.**

- **Alerts and WAP Push are both supported.**

- **Most phones support the ability to select an item from a numbered list.**

  Numbers corresponding to the `<onpick>` item precede each menu choice.

3 Navigation Guidelines

To navigate Wireless Markup Language (WML) content, the user must move through and between cards in one or more decks. The cards can contain many different types of elements, including selection lists (items in a list), displayed information (such as an email message), input fields, multiple-selection lists, buttons, radio buttons, tables, and pop-up menus.

To make applications run on multiple browsers, follow these general rules.

- **Do not assign more than one action of** `<do type="accept">` **in any card.**

- **Always define an action for** `<do type="accept">`**.**

  When no widgets are selected, the accept label is rendered as the softkey label. If no action is defined, a task of `<do type="prev">` is automatically bound to the `<do type="accept">` key with a Back label.

- **Map the most commonly chosen action or most intuitive task to** `<do type="accept">`**.**

- **Create an intuitive label for every task except** `<prev>`**.**

  Assign a `title` attribute to each `<option>` element to provide a descriptive softkey label.

  Example 3-1

  ```
  <option title="Buy">
      <onevent type="onpick">
          Buy tickets
      </onevent>
  </option>
  ```

  In Example 3-1, the label for the primary softkey is Buy for the “Buy tickets” item.

- **Do not create a** `<do type="options">` **with a label of Back or a task of** `<prev/>`**.**

  Creating a `<prev>` task creates a Back on the secondary softkey; however, when no elements are selected on the display, the softkey automatically displays the `prev` action. Thus two Back softkeys are displayed when no `<do type="accept">` task is defined for the card.

- **Capitalize only the first letter of labels of `<do>` elements except in words like OK.**

  Consistent style throughout all applications enhances usability.

- **(GUI) Do not underline text (`<u>` markup tag), because the user may think that the item is a link.**

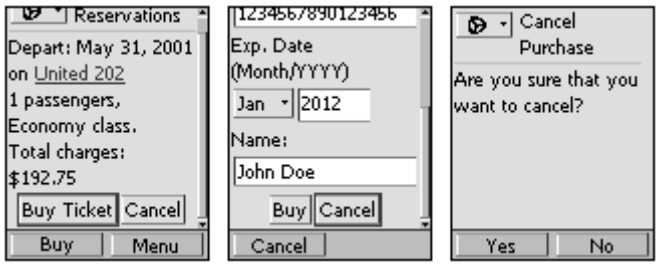
- **Do not bind an action of `<noop/>` to a `<do type="prev">` task.**

  This forces the user to return to the home deck, which is not always intuitive and may make users follow a long path to return to the application. Instead, bind the `prev` task to an intuitive place in the application, to a starting point in the application, or to the home deck.

- **Provide a confirmation card (delete shield) to prevent data loss.**

  Do not allow users to back out of an application inadvertently when they have already entered data in an entry query. Create a card asking users to confirm that they want to quit. This spares users the tedious task of reentering data.

  Example 3-2




```
<card id="cancel" title="Cancel Purchase">
    <p>
        <do type="accept" label="Yes">
            <throw name="bail"/>
        </do>
        <do type="options" label="No">
            <prev/>
        </do>
        Are you sure that you want to cancel?
    </p>
</card>
```

In Example 3-2, the code should be placed on the card at the end of the form. If the user chooses to cancel, the delete shield confirms the action so that entered data is not accidentally lost.

- **All cards should have a `title` attribute of 15 characters or fewer.**

  Titles longer than one line may be truncated, obscuring the meaning. Test the title to make sure that it is displayed appropriately on targeted devices.

  Example 3-3

  ```
  <card id="welcome" title="Welcome to White Pages">
  ```

  In Example 3-3, the last few characters of the title may be truncated.

- **Whenever a phone number is displayed, make it possible for the user to call it.**

  Assign the action `href="wtai://wp/mc;<phone number>"` to create calls, and assign the Call label.

  Example 3-4


```
<card title="United Airlines" id="main">
  <do type="options" label="Done">
    Figure 1
    <prev/>
  </do>
  <p>
    Contact Numbers<br/>
    Milage Plus <a title="call" href="wtai:/wp/mc;18004214655">800.421.4655</a><br/>
    Reservations <a title="call" href="wtai:/wp/mc;1800241.6522">800.241.6522</a><br/>
    Shuttle by United <a title="call" href="wtai:/wp/mc;18007488853">
      800-Shuttle</a><br/>
  </p>
</card>
```

  In Example 3-4, the user can call 1-800-421-4655 by pressing the Call softkey.

- **Whenever possible, keep the order of menu items or forms the same.**

  Users may become familiar with the order and select items without paying much attention.

- **Do not rely on font properties to convey added information.**

  Many phones do not support various font properties such as bold, underline, and italic.

- **Whenever possible, limit the number of softkey actions to two or fewer.**

  This allows the user to view both softkeys and simplifies tasks because the softkey labels are accessible with one keypress. When more than two elements are defined, the first element is bound to the primary softkey, and all other options and elements are accessible from a pop-up menu on the secondary softkey.

  Example 3-5

```
<do type="options" label="Delete">
    <go href="delete.wml"/>
</do>
<do type="options" label="Reply">
    <go href="reply.wml/>
</do>
<do type="options" label="Forward">
    <go href="forward.wml/>
</do>
<do type="options" label="Save">
    <go href="save.wml"/>
</do>
<do type="options" label="Change folder">
    <go href="changefldr.wml/>
</do>
<do type="options" label="Create new">
    <go href="newmsg.wml/>
</do>
```

  In Example 3-5, more than one option is bound to the `<do type="options">` task. In this case, the browser renders a pop-up menu on the secondary softkey with the following options: Delete, Reply, Forward, Save, Change Folder, and Create New.

- **Limit the length of labels that appear on a softkey.**

  The screen size on many phones is limited. Define `<do>` and `<option>` elements and anchor labels using 5 characters or fewer if the labels will be displayed over the softkey: Longer labels may be truncated and lose meaning. Limit the length of `<option>` labels to 13 characters or fewer if two or more `<option>` elements are defined, because these labels can use nearly the full screen when displayed in the pop-up menu.

  Example 3-6

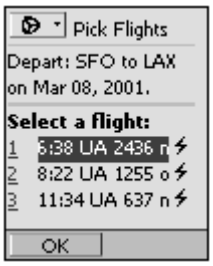
```
<do type="accept" label="Find">
    <go href="find.wml"/>
</do>
```

  In Example 3-6, the Find label is under 5 characters.

- **If the title is too general for the card, include a descriptive header of 15 characters or fewer.**

  This header informs the user of the context when a short title is not descriptive.

  Example 3-7

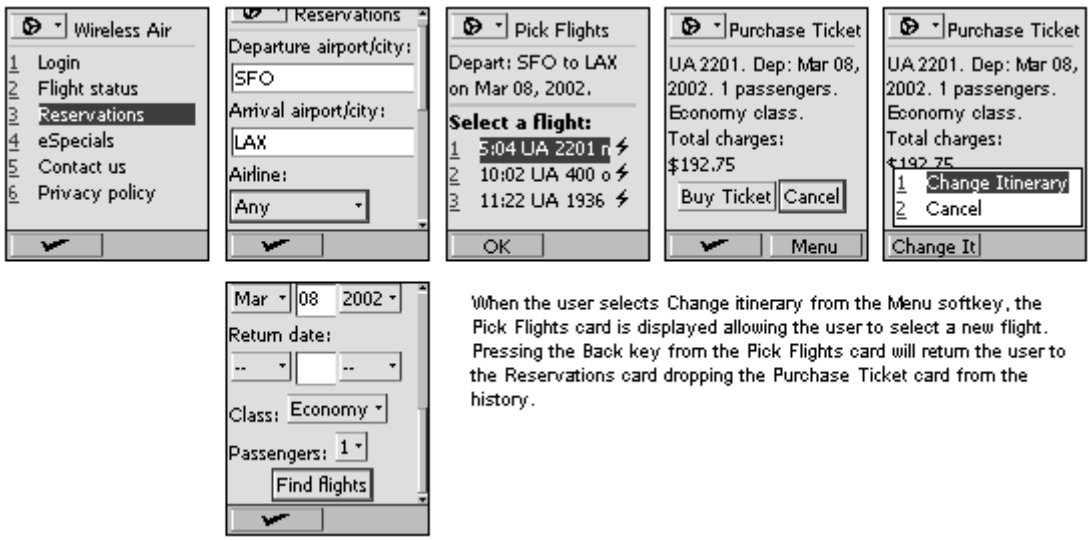


  In Example 3-7, the header “Select a flight” orients the user to the context.

- **Use contexts to direct the Back key to the most intuitive page.**

  Contexts create a start point that allows the application to return to a specified card when the user presses the Back key. The intermediate cards (those accessed before the return to the specified card) are removed from the history and the cache.

  Example 3-8



  In Example 3-8, the goal is to allow the user to navigate backward through the ticket-buying process without losing the flight information that he or she may want to maintain. In addition, the application should ensure that none of the user’s sensitive data remains in the device any longer than necessary. Therefore the application uses contexts to ensure that the cards and variables used to actually purchase the ticket do not remain in the history stack. The sequence of cards in Example 3-8 actually consists of four different WML decks.

Deck1: airres.wml

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.3//EN"
    "http://www.openwave.com/dtd/wml13.dtd">

<wml>
    <card id="Air" title="Reservations">
        <p>
            <do type="accept" label="Find">
                <spawn href="times.wmls#getTime('$airline')">
                    <catch/>
                </spawn>
            </do>
            Departure airport/city:<input name="depart"/><br/>
            Arrival airport/city:<input name="arrive"/><br/>
            Airline:<select type="popup" name="airline" title="Pick Airline">
                <option value="Any">Any</option>
                <option value="AA">American</option>
                <option value="AW">America West</option>
                <option value="AS">Alaska</option>
                <option value="CO">Continental</option>
                <option value="DL">Delta</option>
                <option value="NW">Northwest</option>
                <option value="SW">Southwest</option>
                <option value="TW">TWA</option>
                <option value="UA">United</option>
                <option value="US">USAir</option>
            </select><br/>
            Depart date:
                <do type="button">
                    <go href="calendar.wml">
                        <setvar name="date" value=""/>
                    </go>
                    <img localsrc="calendar" src="" alt=""/>
                </do>
            <br/>
            <select type="popup" name="dmonth" value="Aug">
                <option value="Jan">Jan</option>
                <option value="Feb">Feb</option>
                <option value="Mar">Mar</option>
                <option value="Apr">Apr</option>
                <option value="May">May</option>
                <option value="Jun">Jun</option>
                <option value="Jul">Jul</option>
                <option value="Aug">Aug</option>
                <option value="Sep">Sep</option>
                <option value="Oct">Oct</option>
                <option value="Nov">Nov</option>
                <option value="Dec">Dec</option>
            </select>
            <input name="dday" size="2" format="NN" value="08"/>
            <select type="popup" name="dyear" value="2001">
                <option value="2001">2001</option>
                <option value="2002">2002</option>
            </select><br/>
```

```
            Return date:<br/>
            <select type="popup" name="rmonth" value="--">
                <option value="--">--</option>
                <option value="Jan">Jan</option>
                <option value="Feb">Feb</option>
                <option value="Mar">Mar</option>
                <option value="Apr">Apr</option>
                <option value="May">May</option>
                <option value="Jun">Jun</option>
                <option value="Jul">Jul</option>
                <option value="Aug">Aug</option>
                <option value="Sep">Sep</option>
                <option value="Oct">Oct</option>
                <option value="Nov">Nov</option>
                <option value="Dec">Dec</option>
            </select>
            <input name="rday" size="2" format="*N" maxlength="2"/>
            <select type="popup" name="ryear" value="--">
                <option value="--">--</option>
                <option value="2001">2001</option>
                <option value="2002">2002</option>
            </select><br/>
            Class: <select type="popup" name="class" value="Economy">
                <option value="Economy">Economy</option>
                <option value="Buisness">Business</option>
                <option value="First">First</option>
            </select><br/>
            Passengers: <select type="popup" name="passnum" value="1">
                <option value="1">1</option>
                <option value="2">2</option>
                <option value="3">3</option>
                <option value="4">4</option>
                <option value="5">5</option>
            </select><br/>
        </p>
        <p align="center">
            <do type="button" label="Find flights">
                <spawn href="times.wmls#getTime('$airline')">
                    <catch/>
                </spawn>
            </do>
        </p>
    </card>
</wml>
```

Deck2: srchfltow.wml

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.3//EN"
                "http://www.openwave.com/dtd/wml13.dtd">

<wml>
    <card id="fltresults" title="Pick Flights">
        <do type="accept" label="OK">
            <spawn href="reserveflto.wml">
                <setvar name="depart" value="$depart"/>
                <setvar name="arrive" value="$arrive"/>
                <setvar name="dday" value="$dday"/>
                <setvar name="dyear" value="$dyear"/>
                <setvar name="airline" value="$airline"/>
                <setvar name="dpickflt" value="$dpickflt"/>
                <setvar name="passnum" value="$passnum"/>
                <setvar name="class" value="$class"/>
                <catch name="it"/>
                <catch name="bail">
                    <onevent type="onthrow">
                        <prev/>
                    </onevent>
                </catch>
            </spawn>
        </do>
        <p>
            Depart: $depart to $arrive on $dmonth $dday, $dyear.
        </p>
        <p mode="nowrap">
            <hr size="2"/>
            <b>Select a flight:</b><br/>
            <select name="dpickflt" title="Srchdprt" value="$dfltno1">
                <option value="$dfltno1" title="OK"> $dtime1 $airline $dfltno1
                    non-stop</option>
                <option value="$dfltno2" title="OK"> $dtime2 $airline $dfltno2
                    one-stop</option>
                <option value="$dfltno3" title="OK"> $dtime3 $airline $dfltno3
                    non-stop</option>
            </select>
        </p>
    </card>
</wml>
```

Deck3: reserveflto.wml

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.3//EN"
                       "http://www.openwave.com/dtd/wml13.dtd">

<wml>
    <card id="Air" title="Purcahse">
        <p>
            <do type="accept" label="Buy">
                <go href="#next"/>
            </do>
            <do type="options" label="Change itinerary">
                <prev/>
            </do>
            <do type="options" label="Cancel">
                <throw name="bail"/>
            </do>
            Depart: $dmonth $dday, $dyear on <a href="fltinfo.wml"
                title="Info">$airline $dpickflt</a><br/>
            $passnum passengers, $class class.<br/>
            Total charges: $$192.75
        </p>
        <p align="center">
            <do type="button" label="Buy Ticket">
                <spawn href="#next">
                    <catch name="it">
                        <onevent type="onthrow">
                            <prev/>
                        </onevent>
                    </catch>
                    <catch name="bail">
                        <onevent type="onthrow">
                            <throw name="bail"/>
                        </onevent>
                    </catch>
                    <catch/>
                </spawn>
            </do>
            <do type="button" label="Cancel">
                <throw name="bail"/>
            </do>
        </p>
    </card>
```

```
<card id="next" title="Ticket Purchase">
    <p>
        Credit card:<br/>
        <select type="popup" name="cctype" value="Visa">
            <option title="OK" value="Visa">Visa</option>
            <option title="OK" value="AMEX">American Express</option>
            <option title="OK" value="Mastercard">Mastercard</option>
        </select><br/>
        Card #:<input name="cnum" emptyok="false" format="16N"/><br/>
        Exp. Date (Month/YYYY)<br/><select type="popup" name="month">
            <option value="Jan">Jan</option>
            <option value="Feb">Feb</option>
            <option value="Mar">Mar</option>
            <option value="Apr">Apr</option>
            <option value="May">May</option>
            <option value="Jun">Jun</option>
            <option value="Jul">Jul</option>
            <option value="Aug">Aug</option>
            <option value="Sep">Sep</option>
            <option value="Oct">Oct</option>
            <option value="Nov">Nov</option>
            <option value="Dec">Dec</option>
        </select>
        <input size="4" name="year" value="20"/><br/>
        Name:<input name="cname" emptyok="false"/><br/>
    </p>
    <p align="center">
        <do type="accept" label="Buy">
            <go href="tixconf.wml"/>
        </do>
        <do type="options" label="Change itinerary">
            <throw name="it"/>
        </do>
        <do type="options" label="Cancel">
            <go href="#cancel"/>
        </do>
        <do type="button" label="Buy">
            <go href="tixconf.wml"/>
        </do>
        <do type="button" label="Cancel">
            <go href="#cancel"/>
        </do>
    </p>
</card>

<card id="cancel" title="Cancel Purchase">
    <p>
        <do type="accept" label="Yes">
            <throw name="bail"/>
        </do>
        <do type="options" label="No">
            <prev/>
        </do>
        Are you sure that you want to cancel?
    </p>
</card>
```

```
    <card id="bail" newcontext="true">
        <onevent type="onenterforward">
            <go href="travel2.wml"/>
        </onevent>
    </card>
</wml>

    <card id="next" title="Ticket Purchase">
        <p>
            Credit card:<br/>
            <select type="popup" name="cctype" value="Visa">
                <option value="Visa">Visa</option>
                <option value="AMEX">American Express</option>
                <option value="Mastercard">Mastercard</option>
            </select><br/>
            Card #:<input name="cnum" emptyok="false" format="16N"/><br/>
            Exp. Date (Month/YYYY)<br/>
            <select type="popup" name="month">
                <option value="Jan">Jan</option>
                <option value="Feb">Feb</option>
                <option value="Mar">Mar</option>
                <option value="Apr">Apr</option>
                <option value="May">May</option>
                <option value="Jun">Jun</option>
                <option value="Jul">Jul</option>
                <option value="Aug">Aug</option>
                <option value="Sep">Sep</option>
                <option value="Oct">Oct</option>
                <option value="Nov">Nov</option>
                <option value="Dec">Dec</option>
            </select>
            <input size="4" name="year" value="20"/><br/>
            Name:<input name="cname" emptyok="false"/><br/>
        </p>
        <p align="center">
            <do type="button" label="Buy">
                <go href="tixconf.wml"/>
            </do>
            <do type="button" label="Cancel">
                <go href="#cancel"/>
            </do>
        </p>
    </card>

    <card id="cancel" title="Cancel Purchase">
        <p>
            <do type="accept" label="Yes">
                <throw name="bail"/>
            </do>
            <do type="options" label="No">
                <prev/>
            </do>
            Are you sure that you want to cancel?
        </p>
    </card>
</wml>
```

Deck4: tixconf.wml

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.3//EN"
                       "http://www.openwave.com/dtd/wml13.dtd" >

<wml>
    <card title="Ticket Confirmation" id="ticketconf">
        <do type="accept" label="OK">
            <throw name="bail"/>
        </do>
        <p>
            Conf. #: Q23M5J<br/>
            Depart: $dmonth $dday $dyear on $airline $dpickflt<br/>
            Thank you for using Wireless PCS Travel.
        </p>
    </card>
</wml>
```

- **Make bookmark titles context sensitive.**

  Users should not need to change the title of the bookmark. For instance, instead of using the title “Stock Quote,” use “OPWV Quote.”

- **Make sure that the necessary data is available from a bookmarked item.**

  If the user accesses a bookmarked card, make sure that the appropriate navigation and data are included. For example, if the user bookmarks a stock quote, retain the ticker symbol even though the quote information is updated.

Example 3-9

```
<meta name="vnd.up.bookmark" content="http://stock.com/quote.cgi?OPWV"/>
```

In Example 3-9, when the user bookmarks the site, the label URL is stored regardless of what the URL for the current deck is. This allows for bookmarking the appropriate location of time-sensitive data.

- **Allow users to bookmark entry screens that lead to information.**

  For example, allow users who are retrieving a stock quote to bookmark the entry screen requesting the ticker symbol or company name.

- **Provide an action or link on every card.**

  Every card should define either a link or an action bound to a task of `<do type="accept">`, `<do type="button">`, or `<do type="options">`.

  Example 3-10

```
<do type="accept" label="Find">
    <go href="find.wml"/>
</do>
<do type="options" label="Done">
    <go href="startapp.wml/>
</do>
<p>
    Last name:<br/>
    <input name="lname" title="Last name" size="0"/>
</p>
<p>
    <do type="button" label="Find">
        <go href="find.wml"/>
    </do>
</p>
```

  In Example 3-10, the `<do type="accept">` task calls `find.wml`, and the `<do type="options">` task calls `startapp.wml`. The button following the input field replicates the action of the `<do type="accept">` task so that the user can easily perform the Find function without having to scroll to the end of the card.

- **The WML `help` task renders on the secondary softkey.**

  Defining the WML `help` task renders the help label with the other `<do type="options">` elements on the secondary softkey. This may cause the secondary softkey to display Menu with a pop-up list of `<do type="options">` element as well as the Help item. Therefore, use the WML `help` task only to provide an explanation for a given card rather than for and throughout the entire application.

4 Menu Navigation

Navigating though menus primarily consists of selecting items or links that display a new card or deck or perform some action. Menus can be used for

- Presenting a list of data, such as a list of email messages
- Navigating—for example, choices within a financial application
- Performing an action, such as deleting an email message
- Selecting an option, such as picking the day of the week for a scheduled event
- Changing an option—for example, allowing the user to change a preference or setting

The following guidelines provide information on how to design usable menus for general navigation through your application.

- **Sort items contextually.**

  Items or links should be sorted logically as content dictates: by date, alphabetically, and so on. If there is no logical order, sort by priority; that is, put the item that is most likely to be chosen at the top of the list.

- **Do not put more than nine items on a single card.**

  Limit the amount of scrolling needed on a given card and allow for key shortcuts.

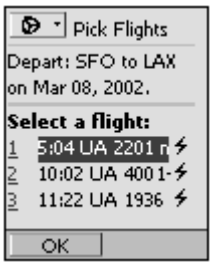
- **If there are more than nine items on a menu, create a More link or a `<select>` element as the ninth item.**

  Clicking More should display the next card of menu items.

- **Do not wrap items on a menu.**

  Use <p mode="nowrap"> to display each menu item on one line. See Example 4-4 for the exception.

  Example 4-1




```
<p mode="nowrap">
  <hr size="2"/>
  <b>Select a flight:</b><br/>
  <select name="dpickflt" title="Srchdprt" value="$dfltno1">
      <option value="$dfltno1" title="OK"> $dtime1 $airline $dfltno1 non-stop</option>
      <option value="$dfltno2" title="OK"> $dtime2 $airline $dfltno2 one-stop</option>
      <option value="$dfltno3" title="OK"> $dtime3 $airline $dfltno3 non-stop</option>
  </select>
</p>
```

  In Example 4-1, <p mode="nowrap"> prevents the menu items from wrapping. Instead, Times Square text scrolling is used.

- **Use the <select> element to get numbers, icons, and items on a menu.**

  Provide quick access to an item by including numbers in a list. Instead of using anchors, use a <select> element for each card. In this case, each item in the list becomes an <option onpick=href> element rather than an anchor.

  Example 4-2

```
<select>
    <option title="Email" onpick="emailmain.cgi">Email</option>
    <option title="Stocks" onpick="stockmain.cgi">Stocks</option>
    <option title="Entertainment" onpick="entmain.cgi">Entertainment</option>
    <option title="Sports" onpick="sportmain.cgi">Sports</option>
    <option title="News" onpick="newsmain.cgi">News</option>
    <option title="Weather" onpick="weatmain.cgi">Weather</option>
    <option title="Travel" onpick="travelmain.cgi">Travel</option>
    <option title="Shopping" onpick="shopmain.cgi">Shopping</option>
    <option title="Weather" onpick="weatmain.cgi">Weather</option>
    <option title="Other" onpick="other.cgi">Other</option>
</select>
```

In Example 4-2, the user can select a menu item by pressing a number key.

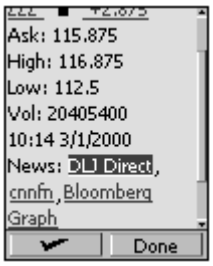
- **(GUI) When links are used, consider assigning the `accesskey` attribute**

  When you define the link with an `accesskey` attribute, the browser numbers the links, so that users can use a shortcut key accelerator.

- **(GUI) It is possible to display links side by side**

  When links are defined sequentially, the default behavior is to list each on a separate line of the display, as in a menu. However, it is possible to display links side by side by adding a punctuation mark (for example, link1, link2) or other text (for example, link1 and link2) between the links. If the links are defined within a `<p mode="nowrap">` block, they are not displayed side by side.

  Example 4-3



```
<card id="quote" title="Stock Quote">
    <onevent type="onenterforward">
        <refresh>
            <setvar name="price" value="+2.875"/>
            <setvar name="icon" value="uparrow2"/>
        </refresh>
    </onevent>
    <do type="options" label="Done">
        <exit>
            <send value="$stock"/>
            <send value="$(price:noesc)"/>
            <send value="$icon"/>
        </exit>
    </do>
    <p>
        <br/>
        <a href="nothing.wml" title="News">$stock
            <img src="" localsrc="$icon" alt=""/>+2.875</a><br/>
        Ask: 115.875<br/>
        High: 116.875<br/>
        Low: 112.5<br/>
        Vol: 20405400<br/>
        10:14 3/1/2000<br/>
        News: <a href="nothing.wml">DLJ</a> <a href="nothing.wml">cnnfn</a>
        <a href="nothing.wml">Bloomberg</a>
        <a href="nothing.wml" title="Graph">Graph</a>
        <a href="nothing.wml" title="Add">Add to Portfolio</a>
    </p>
</card>
```

In Example 4-3, the commas are used to delineate the News sites so that they can be displayed side by side.

- **Wrap text that takes up multiple lines or multiple display windows.**

  For a series of items that occupy multiple lines, allow the lines to wrap and use a link to display the next item. For example, in a list of news headlines, display each headline on a card of its own. Use an anchor with the Skip label at the end so that the user can navigate to the next headline. Map the `<do type="accept">` task to a View label so that the user can view the news item. While the user is viewing the headline, map the `<do type="accept">` task to a Skip label, so that the user can skip to the next headline.

  Example 4-4

```
<card id="story1">
  <do type="accept" label="Skip">
    <go href="#story2"/>
  </do>
  <do type="options" label="Done">
    <exit/>
  </do>
  <p>
    Top Stories 1/3<br/>
    Astronomers find evidence of planet in constellation Leo.
    <a href="story1full.wml" title="View">View</a>
    <a href="#story2" title="Skip">Skip</a>
  </p>
</card>

<card id="story2">
  <do type="accept" label="Skip">
    <go href="#story3"/>
  </do>
  <do type="options" label="Done">
    <exit/>
  </do>
  <p>
    Top Stories 2/3<br/>
    Sunspot activity at an all time high.<br/>
    <a href="story2full.wml" title="View">View</a>
    <a href="#story3" title="Skip">Skip</a>
  </p>
</card>

<card id="story3">
  <do type="options" label="Done">
    <exit/>
  </do>
  <p>
    Top Stories 3/3<br/>
    Stock markets worldwide gaining.<br/>
    <a href="story3full.wml" title="View">View</a>
  </p>
</card>
```

In Example 4-4, a list of items with long text is broken into a card with the header information. The user can skip to the next header by pressing the Skip softkey or the Skip link at the end of the text.

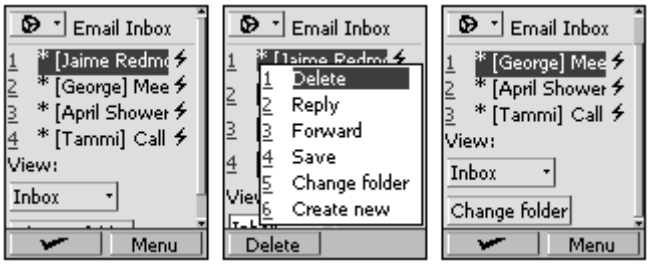
- **An anchor should have a descriptive label of 5 characters or fewer.**

  The label for links leading to the defined URL should be 5 characters or fewer. It is rendered as the softkey label.

- **Allow multiple actions to be performed on a selected item.**

  To allow multiple actions on an menu item, set the `type` attribute of the `select` element to `list`. For example, to allow the user to delete, forward, or reply to an email message without having to view it, assign the `value` attribute for each `<option>`. Make the desired action available via `<do type="options">`, which key off of the value of each item.

  Example 4-5




```
<wml>
    <card title="Email Inbox" id="inbox">
        <onevent type="onenterforward">
            <refresh>
                <setvar name="imgA" value="*"/>
                <setvar name="imgB" value="*"/>
                <setvar name="imgC" value="*"/>
                <setvar name="imgD" value="*"/>
            </refresh>
        </onevent>
        <do type="accept" label="View">
            <spawn href="email/mail$(msg:noesc)">
                <catch/>
            </spawn>
        </do>
        <do type="options" label="Delete">
            <go href="file:c:/opwvdemo/nothing.wmlc"/>
        </do>
        <do type="options" label="Reply">
            <spawn href="#reply$(msg:noesc)">
                <catch/>
            </spawn>
        </do>
        <do type="options" label="Forward">
            <spawn href="emailfwd.wmlc#fwd$(msg:noesc)">
                <catch/>
            </spawn>
        </do>
        <do type="options" label="Save">
            <go href="saved.wmlc#saved$(msg)"/>
        </do>
        <do type="options" label="Change folder">
            <go href="file:c:/opwvdemo/nothing.wmlc"/>
        </do>
```

```
<do type="options" label="Home">
    <go href="#home"/>
</do>
<do type="options" label="Create new">
    <spawn href="emailedit.wmlc">
        <catch/>
    </spawn>
</do>
<p mode="nowrap">
    <select type="list" name="msg">
        <option value="1">
            <onevent type="onpick">
                <spawn href="email1.wmlc">
                    <catch/>
                </spawn>
            </onevent>
            <img localsrc="" alt="" src="../images/paperclip.png"/>
            [Jaime Redmond] Do you want to go to
        </option>
        <option value="2">
            <onevent type="onpick">
                <spawn href="email2.wmlc">
                    <catch/>
                </spawn>
            </onevent>
            <img alt="" src="../images/108.png"/>
            [George] Meeting changed to 3:30
        </option>
        <option value="3">
            <onevent type="onpick">
                <spawn href="email3.wmlc">
                    <catch/>
                </spawn>
            </onevent>
            <img alt="" src="../images/2.png"/>
            [Mette Schand] Conference call with Mette
        </option>
        <option value="4">
            <onevent type="onpick">
                <spawn href="email4.wmlc">
                    <catch/>
                </spawn>
            </onevent>
            <img src="../images/108.png" alt=""/>
            [Edward Graves] Call me when convenient
        </option>
    </select>
    View:
    <select name="fldr" type="popup" value="Inbox">
        <option value="file:c:/opwvdemo/nothing.wmlc">Inbox</option>
        <option value="#outbox">Outbox</option>
        <option value="file:c:/opwvdemo/email/sent.wmlc">Sent</option>
        <option value="../nothing.wmlc">Templates</option>
    </select>
</p>
```

```
    <p align="center">
        <do type="button" label="Change folder">
            <go href="$(fldr:noesc)"/>
        </do>
    </p>
</card>
```

In Example 4-5, the user can delete a message directly by highlighting the message header and choosing Delete from the Menu softkey. The code in the menu card is vastly simplified to show how the menu items would be rendered on the device. In a real application, the code would be generated dynamically so that the server could be accessed with each keypress. The behavior for each individual message can be controlled from the menu only through this type of interaction with the server.

- **Be aware that links may be displayed differently on different phones.**

  Most phones underline links, and a few may display them in square or angle brackets. Therefore, do not underline text or use brackets in text.

Making Phone Calls from the Browser

5

Some applications require the user to make a call or create opportunities for the user to do so—for example, from a list of contacts, a phone number query, or an order form. This is an extension of general navigation; however, not all phones allow the user to make a call directly from the browser. If the phone does support calls from the browser, develop the application so that the user can retrieve the phone numbers.

- **When displaying more than one phone number on a card, display the primary number first.**

  That way the user sees the most frequently chosen number without having to scroll.

- **If the user exited the browser for a voice call, when the user terminates the call, assume that the phone may reinvoke the browser.**

- **Do not assume that the same deck will be redisplayed after the browser is reinvoked.**

  The user may need to navigate back to the same deck that was displayed before the call.

- **Embed code to make the phone call.**

  Example 5-1


```
<do type="accept" label="Call">
    <go href="wtai://wp/mc;18004214655">
</do>
```

In Example 5-1, the user can call 1-800-421-4655 by pressing the Call softkey. This action can also be bound to a button.

- **Provide a Call anchor label.**

  This allows users to make a call directly from the browser.

- **Use** `href="wtai://wp/mc;<phone number>"` **instead of a link, unless multiple actions are desired.**

  This allows the user to make the call directly from the card rather than accessing another card to retrieve the number and make the call.

- **Expect the browser to return to the previous card when it is reinvoked after the call is terminated.**

  To display a card other than the previous card in the history after the call terminates, use `<onevent type="onenterbackward">` in the card that generated the call.

- **Map the Send key action to the action of calling.**

  In addition to using the Call softkey, use `<do type="vnd.up.send">` so that users can also press the Send key to make the call. However, do not rely solely on this method because not all phones have a Send key or map the Send key function.

Using Multiple-Selection Lists

6

Use multiple-selection lists when the user can select more than one item from a list.

- **Create only one** `<do>` **element or** `<do type="accept">` **label, using 5 characters or fewer, for the multiple-selection list.**

  Only one label is displayed; the second label is reserved for the device.

  Example 6-1


```
<do type="accept" label="Done">
    <go href="getnames.cgi"/>
</do>
<select name="addrbk" multiple="true">
    <option value="ashowers">April Showers</option>
    <option value="bbrown">Bill Brown</option>
    <option value="bjones">Bob J</option>
    <option value="jimsimpson">Jim S</option>
    <option value="joesmith">Joe Smith</option>
    <option value="salto">Sal</option>
    <option value="tjohnson">Tammi Johnson </option>
</select>
```

  Example 6-1 demonstrates how to create a multiple-selection list. Notice that the code does not use `<do type="options">`, because it would render a Menu softkey displaying the defined and `prev` actions.

- **Do not use** `<option onpick=href>` **for items on the selection list.**

  Using `<option onpick>` causes the browser to access the associated URL before giving the user the chance to select the desired items.

- **Use** `<p mode="nowrap">` **to allow long text to Times Square scroll.**

  Try to fit the text on one line so that the text does not scroll.

- **Use multiple choice to minimize typing.**

  For applications that have specific choices, allow users to choose from a multiple-selection list. It is also possible to use a database on a server to access possible input choices. For example, use an email address book to allow users to insert email addresses into the recipient line of an email message.

7 Backward Navigation

Backward navigation is a key to a usable application. Pay special attention to backward navigation because users tend to use the Back key or softkey to back out of an application. Users are more likely to trust applications with good backward navigation functions. Also, backward navigation lets users leave the application without returning to the home deck. This can be particularly helpful for applications that are deeply embedded in the browser. For example, if a set of applications is three menus deep from the home card, backward navigation allows the user to return easily to the start of the set without renavigating through the first two or three menus. In some cases, the design must prohibit navigation behind password-protected cards.

- **(GUI) Never define a `<do type="options">` with a label of Back.**

  When no `<do type="accept">` is defined, the browser automatically displays a Back softkey with a `<type="prev">` task. When a `<do type="prev">` task is used, the phone may display two Back softkeys, which can confuse the user.

- **When direct backward navigation is not suitable, map backward navigation to the next highest or most intuitive menu.**

  Backward navigation is not always practical. For example, suppose that the user confirms an action, such as making a purchase or deleting an item. Pressing the Back key should not take the user to the card confirming the purchase or to the list with the deleted item. Instead, the application should return either to the application's home card, to a login card, or to a card that lets the user continue through the application intuitively or exit easily. Use an `<onevent type="onenterbackward">` task either to prevent backward navigation or to take the users past screens that they should not revisit.

  Example 7-1

  ```
  <onevent type="onenterbackward">
      <prev/>
  </onevent>
  ```

  Example 7-1 shows how to prevent the user from navigating back through a card in a password-protected area. In this case, the user would be taken back an additional card.

- **(GUI) When the action of the Back key is overridden, be sure to override the Accept key with the same action when no other `accept` action is present on the card.**

  If no action is bound to the Accept softkey, the key is assigned an action of `prev` with a label of Back. If the backward navigation path for the Prev key has been modified, the `back` action on the primary softkey will be different from the action on the physical Back key. This is likely to confuse the user, so be sure to mirror the action bound to the physical Back key on the primary soft key.

```
<do type="prev">
    <exit/>
</do>
<do type="accept" label="Back">
    <exit/>
</do>
```

- **Create a second way to navigate backward when moving back in the history stack is not desirable.**

  If backward navigation should return the user to a higher menu or to the top menu of the application, provide backward navigation via `<do type="options">`. Use the Done label to indicate that backward navigation will take the user back more than one step. The Done label appears on the secondary softkey.

  Example 7-2


```
<do type="options" label="Done">
    <go href="appmain.wml"/>
</do>
```

  Example 7-2 shows how a URL helps the user navigate more easily. The Done and Back items take the user back to the application's home card.

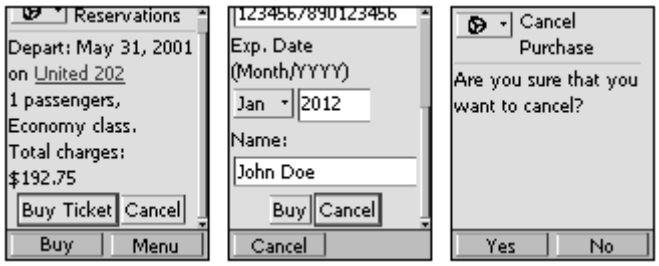
- **Save the values of variables when necessary.**

  If the user exits an entry field, it may be helpful to temporarily save the values of all or some of the variables. This can reduce the amount of information the user must enter in the future. For example, it may be helpful to retain nonsecure information entered in an order form, such as the name and address, so that the user does not have to reenter it in the same browsing session. See Chapter 15, "Cookies," for information about using cookies for this purpose.

- **Provide delete shields.**

  If backing out of an application may cause data loss, provide a card asking the user to confirm the action.

  Example 7-3

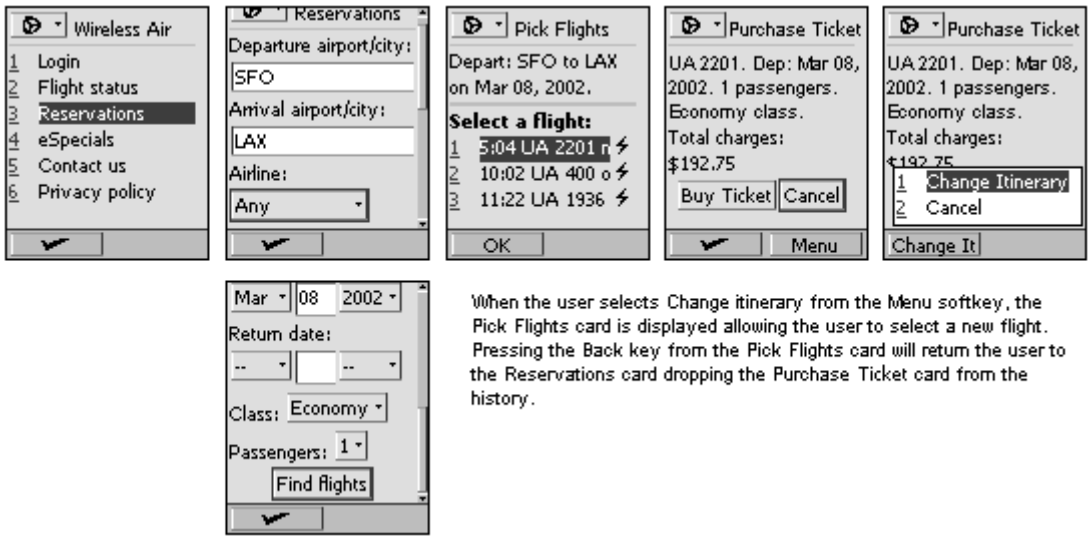


  In Example 7-3, the confirmation card acts as a delete shield, preventing loss of data if the user backs out of the card with the Name prompt. See Example 3-8 for a complete description of the code.

- **Use contexts when necessary.**

  That way, the user does not have to repeatedly press the Back or Clear key, clearing already entered data.

  Example 7-4



  Example 7-4 shows the navigation of contexts. See Example 3-8 for a complete description of the code.

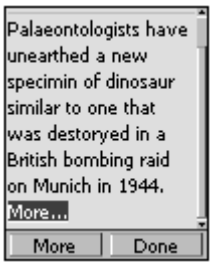
8 Displaying Text

Some cards contain mostly text. For example, applications that display email messages, news items, stock quotes, and confirmation or informative notices are text intensive. These cards often contain a limited number of selection choices and are not used for text entry.

- **Display about 500 to 800 characters per card.**

- **Define a More link if more information is available.**

  Example 8-1



- **Wrap the text.**

  Do not use `<p mode="nowrap">`. The exception is a short header or text of little relevance to the user. For example, when displaying news for a specific company, use the `<p mode="nowrap">` mode to display the company name.

- **Define the primary label for navigation.**

  Use the primary softkey for forward navigation (accessing the next set of data); use the secondary labels for alternative navigation or other functions related to the application.

- **Define a Skip link to go to next related item.**

  When displaying a series of related data, such as news stories or email messages, use a Skip link to allow the user to skip the current item and retrieve the next one. Do not use Next. Usability tests show that users tend to think that Next means “go to the next page,” not “go to the next item.”

- **Define labels for links.**

  Create an appropriate label that reflects the action of the link. When the user selects the link, the label should change accordingly—for example, from View to Skip. Limit labels to 5 characters.

- **Use links sparingly.**

  It is popular to put links at the end of display cards for alternative navigation, but do not define more than two or three links per card. User tests show that links at the end of the card often make it difficult for users to navigate out of the card. The reason is that labels corresponding to the link replace the labels for the primary action. This forces the user to scroll back up to access the primary action. The last link should match the default action for the card so that the user does not have to scroll up again.

- **Keep the text of links short.**

  This avoids links that wrap beyond one line.

- **Place navigation links only at the top and bottom of the card.**

  Do not embed navigation links in displayed text (unless it is context sensitive), because users will not understand that the links are not related to the data. Limit the length of navigation links to one line each, 13 characters if possible (the brackets use 2 of the 15 characters).

Example 8-2

- **Use mobile-originated prefetch to access the next card.**

  This shortens the time needed to retrieve the next set of information. While the user is reading one card, the next card can be retrieved and put into the cache. Be sure that the information is cache does not expire before the user is likely to access it.

  Example 8-3

```
<head>
    <link href="page2.wml" rel="next/>
</head>
<card id="page1">
    <do type="accept" label="Page2">
        <go href="page2.wml"/>
    </do>
    <p>
        Page 1 of 2<br/>
        ...
    </p>
</card>
```

  Example 8-3 shows how to use prefetch to access the next card in a deck.

- **Do not use links on cards used to display results.**

  On cards that ask the user to confirm an action (for instance, deleting a contact from an address book), that provide error explanations, or that display other results, use the `<do type="accept">` and `<do type="options">` labels instead of links. Map the safest or most common response to the `<do type="accept">` label.

- **Incorporate Done softkeys when possible.**

  If no secondary functions on the displayed text are needed, add a Done softkey that returns the user to the next highest level within the application. This is often used in conjunction with contexts so that the user can pop out of the current context and return to the context that spawned it.

9 Data Entry Queries

If the user must enter numeric or alphanumeric information, the application can use entry queries to elicit it.

- **Minimize the number of input fields that require alphanumeric entry.**

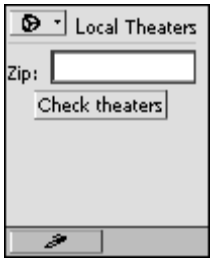
  It is difficult and time-consuming to enter text on mobile phones. Whenever possible, use selection lists, pop-up menus, or option buttons to avoid data entry. Another strategy is to store the user's entries and reuse them when appropriate.

- **When an input field is activated by the user, no action or label can be defined for that field.**

  Use only one `<do type="accept">` task. Additional options hide primary navigation from the user.

- **Define a descriptive `title` attribute and use descriptive text of 15 characters or fewer for all `<input>` elements.**

  Example 9-1




```
<card id="theaters" title="Local Theaters">
    <do type="accept" label="OK">
        <go href="theaters.cgi"/>
    </do>
    <p>
        Zip:
        <input name="zip" title="Zip" format="NNNNN"/>
    </p>
</card>
```

  Example 9-1 shows how to create a card that requests data. Note that only one action label can be defined.

- **Limit `<input>` elements to 254 characters.**

  Use the `maxlength` attribute to restrict the length.

- **(GUI) To make a text field growable, do not define the `size` attribute**

  When no `size` attribute is defined, the text field box increases as the user types. This is helpful for fields that can be an undetermined length, and prevents users from having to scroll through an empty text box. Use the `maxlength` attribute rather than the `size` attribute to restrict the numbers of characters that can be entered.

- **(GUI) To fix the size of the text box to one line, use the `size="0"` attribute.**

  Use the `size="0"` attribute only when the user does not need to view all of the text in the field—for example, a list of recipients in an email application. In addition, make sure that enough characters are visible to the user, particularly if the input field follows text, because the input element is rendered on the same line of the display as any text that precedes it. If you do not know whether the field is large enough to support enough characters, add a line break after the displayed text and before the input field.

- **(GUI) To fix the size of the input field, define the `size` attribute to be other than `size="0"`.**

  Defining the `size` attribute limits the displayed size of the field but does not limit the amount of data a user can enter. The value of the `size` attribute specifies how many characters the browser displays to the user. If it is also necessary to restrict the number of characters, define the `maxlength` attribute. If the size of the field does not fix on the line with additional text, the field automatically starts on the next line.

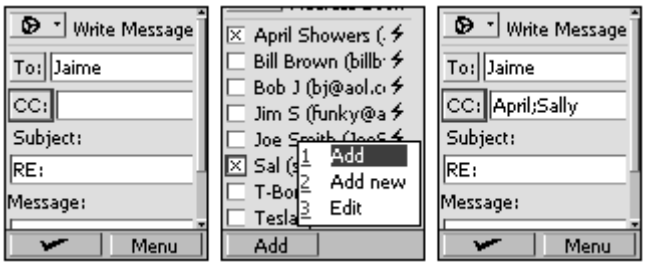
- **(GUI) If the `size` attribute is defined to be greater than the length of one line (in characters), the input field displays the entire line.**

  If the `size` attribute is defined to display over multiple lines—for example, `size="20"`—the text box is drawn to display two full lines rather than only 20 characters.

- **(GUI) Use a button or a link after the final input field to allow the user to submit the data.**

  The user should be able to submit the data easily by pressing a link or a button. Buttons are useful in forms, but they take up more space than links. Define the `<do type="accept">` element to replicate the action on the button or link.

  Example 9-2



From emailedit.wml:

```
<do type="button" label="To:">
    <spawn href="emailaddr.wmlc">
        <receive name="moreto"/>
        <catch/>
    </spawn>
</do>
<input size="0" name="newfro"/>
<do type="button" label="CC:">
    <spawn href="emailaddr.wmlc">
        <receive name="morecc"/>
        <catch/>
    </spawn>
</do>
<input name="cc" size="0" value="$morecc"/>
```

From emailaddr.wml:

```
<select name="addresses" multiple="true">
    <option value="April">April Sanders (as@wet.com)</option>
    <option value="Bill Brown">Bill Brown
        (billby@yahoo.com)</option>
    <option value="Bob Jones">Bob J (bj@aol.com)</option>
    <option value="Jimmy">Jim S (funky@aol.com)</option>
    <option value="Joe Smith">Joe Smith
        (JoeS@funtv.com)</option>
    <option value="Sally">Sal (sal@mail.com)</option>
    <option value="Tad Branson">Tad Brason
        (tb@excite.com)</option>
    <option value="Tesla">Tesla (coolkat@hotmail.com)</option>
</select>
<do type="button" label="Add">
    <exit>
        <send value="$addresses"/>
    </exit>
```

In Example 9-2, To: and CC: are buttons that allow the user to select email addresses from a contact list and insert them into their respective fields.

- **Include a descriptive label for the `<do type="accept">` task.**

  For example, use the Find label in applications that allow the user to access information from a database. This should replicate the action on a button or link for submitting the form.

- **(GUI) Use a button to allow users to insert data in a text query.**

  Provide a button that allows users to automatically insert text into a text query. For example, to allow users to pick a name from an address book, provide a button that links to the address book. When the user selects a name, the input field is automatically filled.

- **Whenever possible, make password fields numeric only.**

  It is easier to enter numbers than letters or symbols.

- **Do not mask alphanumeric passwords.**

  It is easier for the user to hide the display from others than to type with masked characters.

- **Discourage long passwords by using `maxlength` to restrict the length of the input.**

  Set `maxlength` to limit the length of user names to no more than 32 characters and passwords to no more than 20 characters. If there is an associated web application, make sure that `maxlength` is long enough that the user can access account information from the WAP application.

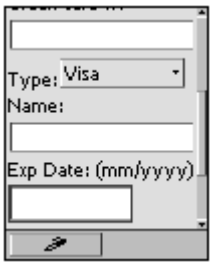
10 Formatted Entry Fields

In some applications, the data entry queries can provide specialized format fields that guide the user in entering the required information. For example, if the user must enter a credit card number of 16 digits, the entry field can be formatted to accept exactly 16 characters. Other formatting is also possible; for example, the browser can be limited to accept only numeric entries.

- **Create informative titles.**

  For example, if the field requires a year in a specific format, the field title should indicate what is required (yyyy).

  Example 10-1




```
Exp Date: (mm/yyyy) <br/>
     <input name="expdate" format="NN\/NNNN" emptyok="false" size="7"/><br/>
```

  Example 10-1 shows how to create an input field that requests formatted data. The field accepts only numeric input no longer than 6 digits. The label also informs the user what input is expected.

- **Force the entry type to numeric or alpha, if appropriate.**

  For example, in some environments, postal codes are numeric only. For those countries, force the entry type to numeric so that the user cannot enter alphabetic characters.

- **Use** `maxlength` **to restrict the length of the string, if required.**

  This is helpful for phone numbers or credit card entries.

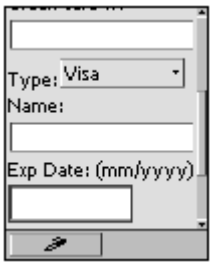
- **Prefill known data.**

  For example, the first 2 digits of the year (20xx) can be automatically filled for the user. The user can never change prefilled information, so be careful not to prefill any information that is subject to change.

- **Use `emptyok="false"` to prohibit users from continuing without entering data.**

  Prohibit users from submitting forms that lack required text by using the `emptyok="false"` attribute.

  Example 10-2




```
<input name="ccnum" emptyok="false" format="NNNNNNNNNNNNNNNN"/><br/>
    Type:
    <select type="popup" name="cctype">
        <option value="Visa">Visa</option>
        <option value="American Express">AMEX</option>
        <option value="Master Card">Master Card</option>
    </select><br/>
    Name:<br/>
    <input name="ccname" emptyok="false"/><br/>
    Exp Date: (mm/yyyy) <br/>
    <input name="expdate" format="NN\/\2\0NN" emptyok="false" size="7"/><br/>
```

  Example 10-2 shows how to add a slash automatically after the user enters the first 2 digits.

- **If the entry typically is numeric or starts with a number, set the default input mode to numeric but also allow the user to enter letters or symbols.**

  For example, phone numbers generally consist of numbers only, but in some cases the user may also need to enter a special character, such as + or #. This is also true for postal codes for some countries, purchase orders, ticket confirmations, and part numbers.

  Example 10-3

  ```
  <input name="phnum" type="phonenum">
  ```

  Example 10-3 shows how to use `<type="phonenum">` so that the input mode is initially numeric but users can change to alphanumeric or symbol mode.

- **Include symbols in the format to simplify text entry.**

  Use symbols (such as / : ; -) in the formatting string to provide visual information that indicates the expected input. Example 10-2 shows how to embed special characters in the input field.

- **When using the format `nN` or `nM`, keep in mind that the user can enter from 0 to n digits or characters.**

- **Set the correct case (lowercase or uppercase).**

  Example 10-4

  ```
  State (2 Letter Code)
      <input name="state" format="MM"/>
  ```

  Example 10-4 shows how to set the appropriate case when the user first accesses the input field.

- **The default is M*m formatting, which results in sentence-like formatting.**

11 Forms

Forms can encompass one or more display or action types on a card or set of cards. There are two primary types of forms: forms that allow the user to enter information sequentially (wizard forms) and forms that show a combination of elements in one list.

- **(GUI) Multiple elements can be displayed on a single card.**

  It is possible for a variety of elements to be displayed on a single card. For example, it is possible to put two input elements on the same card to request login and password information, or a `<select type="popup">` and input element to request the date.

- **(GUI) Never define a `<select type="list">` element to set a value in a form.**

  Use `<select type="radio">` (for four items or fewer) or a `<select type="popup">` (for five items or more) instead of using a `<select type="list">` element.

- **(GUI) Whenever possible, use wizard forms to break the data into screen-size portions of related information.**

  For example, for purchases, create one card for selecting the item, another for the billing information, and a third for shipping information or other preferences. Do not put all of the information on one card, because the card can become too long to navigate easily.

- **Link cards in a logical order.**

  Provide a logical order that suggests what needs to be entered or selected and why.

- **Include descriptive text and titles for all `<input>` and `<select>` elements.**

- **(GUI) After the final element, add a button that goes to the next card in the sequence or submits the data.**

  For example, if the card is a contact search form, add a Search button that allows the user to search on the entered or selected fields. For longer forms, add a second button, labeled Cancel, that allows the user to exit the application. Retain the data, but use contexts to drop the cards in the history.

- **(GUI) Include a descriptive label for the `<do type="accept">` task.**

  For example, use the Find label in applications that allow the user to access information from a database. This should replicate the action on a button or link for submitting the form.

- **(GUI) Do not place a button in the middle of a form unless it is used to insert a text field.**

  Users often select the button without looking to see if there is additional information, particularly because a button requires more space to display than normal text. However, a button can be used to insert text in a field. See the information on inserting text in a field in Chapter 9, “Data Entry Queries.”

- **(GUI) Use a link in the middle of the form only if the additional items on the card are not required.**

  Because users may select a link without looking to see if there is additional information, you should use a link only if no other text or elements below the link are essential.

- **(GUI) If the user does not need to select or enter many elements on the form, define a `<do type="options">` task to allow the user to access the next card.**

  Bind the action of `<do type="options">` to the next card and define a second `<do type="options">` with the Cancel label that allows the user to exit the application.

- **The primary softkey label should be Next for all cards in the sequence except the final card.**

  The label for the final card in the wizard should explain the final action, such as Save, Send, Order, or Buy. Limit the label to 5 characters.

- **Create a final verification or confirmation card that displays all entered or selected values.**

  Allow the user to change the values if necessary.

- **If the user tries to cancel out of a wizard, provide a delete shield.**

  Provide a delete shield to prevent accidental data loss when the user presses the Back key on the first card in a form or Cancel while in a wizard form.

- **Limit the number of characters preceding a `<select>` or `<input>` element to 30.**

  This limits the text to approximately two lines. Long texts that precede the default selected option or input field will scroll off the screen.

- **(GUI) Use option buttons to allow users to pick from a list of 3 or 4 items.**

  Use option buttons to replace a selection list in a form. Limit the number of option buttons to a number that can be displayed on one screen.

- **(GUI) Use pop-up menus to allow users to choose from a list of 5 to 9 items.**

  Limit the number of items in a pop-up menu to approximately 5 to 9 items, or to a reasonable number of items from a logical list, such as a list of the 12 months.

- **(GUI) Use multiple-selection lists to allow users to choose multiple items or to set the state of an item to on or off.**

- **Provide an appropriate label for the primary action.**

  Label the `<do type="accept">` task with the desired result. For example, in a search query, define the `<do type="accept">` label as Find.

- **Display a final card that shows the selected or entered results for all fields.**

  When the application saves user data for future access, provide a card displaying the data entered and the elements on the form. For example, in an address book application, show the user all of the data entered for that contact.

12 Alerts

Alerts are a feature supported on the Openwave Mobile Access Gateway and the Openwave Mobile Browser. They add value to an application by notifying users of new information in their areas of interest.

The Openwave Mobile Browser 5.0 supports WAP Push according to the WAP 1.2.1 specifications. For information about how to deliver a notification to a device using WAP Push, refer to the *Openwave SDK, WAP Edition, Developer's Guide*.

- **Alerts support alert type (priority), time to live (TTL), alert removal, delivery status, and security.**

  Refer to the *Openwave SDK Developer's Guide* for information about how to implement alert notifications.

- **Allow the user to turn off or change the alert setting for the application.**

  This may not be supported on all phones, but whenever possible you should allow users to customize settings according to their preferences.

- **Use only one alert inbox slot, and make sure that the same URL is used for all of the application's alerts.**

- **Give alerts a short title of 15 characters or fewer.**

  The title should fit on one line.

- **Do not include specific data in a title.**

  The title is a way to group messages under one alert heading. The same alert title should be sent whenever messages are delivered from an application. This is the title that appears in the inbox, not the message title. That is, only one alert title appears in the inbox, even if more than one message is sent to that box.

- **Make sure that the URL remains active for at least 24 hours and that the user can access the alert once it is in the inbox.**

- **Alerts are delivered to a device by specifying the subscriber ID in the Openwave Mobile Access Gateway address.**

  The subscriber ID is delivered with an `HTTP_X_UP_SUBNO` header in every HTTP request from a device. The absence of the header in the HTTP request means one of two things: the user is not provisioned (and alerts cannot be sent) or the gateway is not an Openwave gateway.

- **The alert consists of at least the title string, URL, priority, Openwave Mobile Access Gateway address, and subno (subscriber ID).**

13 Icons and Images

Images can enhance or support the displayed information so that the user can quickly review a list of items or see a trend. For example, a weather report can display a date along with an icon of the predicted weather. Similarly, an up or down arrow can precede a stock quote.

- **All phones support the `wbmp` image format.**
- **All phones with color capability also support the `png` format for color images.**

  Use the web-safe color palette to ensure that the color scheme of the image is respected.
- **Always include descriptive `alt` text for devices that do not support images.**
- **If the phone talks to an Openwave Mobile Access Gateway, the server compiles 1-bit `bmp` images to `wbmp` form.**
- **The WAP Forum does not currently define an animated image format.**
- **Do not define associate functions for areas in an image.**

  There is no way to associate an area in a `bmp` image to an action (that is, there is no image map function).
- **Be careful using images on cards with a timer element, because the timer may expire before the image is loaded.**

  This is not an issue if the image is delivered with the deck or card in a digest.
- **Images larger than the display size are scrollable vertically but not horizontally.**

  Make images no wider than 40 pixels.

- **Use preloaded images.**

  Openwave offers `localsrc` images that are preloaded into the devices that support images. The use of these images shortens network access time and creates a consistent user experience. A list of the `localsrc` images is available in the "Images" section of the WML reference that ships with the SDK from Openwave, which can also be accessed at `http://developer.openwave.com`. See the "Image" section in the *WML 1.3 Language Reference* for the `localsrc` images.

- **Images are aligned according to the attribute of the** `<p>` **element.**

- **Images can be displayed inline, along with text or a link.**

- **Images can be included in an** `<option>` **element.**

  This allows an icon to be displayed on the same line as a menu item.

  Example 13-1

```
<option onpick="my_url"><img localsrc="envelope" src="" alt=""/>Email</option>
```

  Example 13-1 shows how to embed an image inline with a menu item. In this case, an envelope is displayed before the text (✉ Email).

- **When delivering a deck that calls images that are not already in the cache, use a digest so that the image is displayed when the card has finished loading.**

  This loads the deck and image simultaneously. The maximum digest size must be less than the `MAX PDU`, which is device specific but is approximately 2000 bytes.

14 Cache

Cache management is important for allowing quick access to previously viewed cards and for controlling the display of time-sensitive content.

- **Do not leave time-sensitive data, such as stock quotes, in the cache.**

- **Use a cache-control directive to specify how long a deck should persist in the cache of a device.**

  For example, a cache-controlled directive can prevent users from accessing outdated time-sensitive information, such as weather, traffic, or a stock quote. Cache-control directives are at the deck level rather than the card level.

  Example 14-1

```
<meta http-equiv="Cache-control" content="max-age=600" forua="true"/>
```

  In Example 14-1, the value 600 represents the number of seconds that the data should be marked as valid in the cache.

- **Do not build an application that relies on information residing in the cache.**

- **Force the reloading of the deck for dynamic data.**

  Use HTTP cache-control with `no-cache` and `must-revalidate` to ensure that a deck is always requested from the server. For example, this can be used in applications that have an “Update information” link to dynamic data.

  Example 14-2

```
<meta http-equiv="Cache-control" content="no-cache" forua="true"/>
<meta http-equiv="Cache-control" content="must-revalidate" forua="true"/>
```

  Example 14-2 shows how to force reloading of the deck every time the card is accessed in the forward or backward direction.

- **The default time to live (TTL) for a deck is 30 days or until memory is exhausted.**

- **If the user is likely to access the next card, allow the browser to prefetch the next deck.**

  For applications providing textual information that users are likely to access in sequence, such as news or email, use the prefetch API from the Openwave SDK.

  Example 14-3

```
<head>
    <link href="page2.wml" rel="next"/>
</head>
<card id="page1">
    <do type="accept" label="More">
        <go href="page2.wml"/>
    </do>
    <p>
        Page 1 of 2<br/>
        ...
    </p>
</card>
```

  Example 14-3 shows how to prefetch the content of the next card. When the user loads the deck, `page2.wml` is automatically loaded in the background as soon as the current deck has finished loading. This ensures that `page2.wml` is already in the cache when the user presses the `<do type="accept">` key. Do not assign a `max-age` of 0 to data that is prefetched.

15 Cookies

Use cookies to store data, thereby reducing the amount of information the user must enter.

- **Cookies are not stored in the phone.**

  However, the phone can access cookies if it is connected to an Openwave Mobile Access Gateway. If you do not know whether the device will access an Openwave Mobile Access Gateway, the information should be stored on the server where the application resides.

- **Use cookies as needed.**

  Cookies can save the user from continually having to enter data from the keypad. If specific information needs to be stored, provide a login so that the application can validate the user and access that user's personal information. Dynamically configure the login menu item so that it is displayed only if the application cannot identify the user.

- **If session information needs to be retained, use URL rewriting.**

- **Always provide an alternative to using cookies.**

  Make sure that the code supports cases when the gateway/browser combination does not support cookies.

16 Labels and Links

Depending on the type of application and type of information displayed, you should use consistent labels in the application and across other applications. Capitalize only the first letter of the label or link unless the word is always uppercase, such as OK.

To create a label for a softkey, define a `title` attribute for each `<option>` element. If you define a `title` attribute for one element, you should do so consistently throughout the application.

For menu text or text on a softkey, capitalize only the first word rather than each word, except when the menu item represents more than one category; for example, Dates, Places, and People.

Recommended Labels and Links

- **OK: Used to select a menu option in a choice card.**

  Can also signify agreement to an operation, such as sending an email message.

- **Done: Used to allow the user to cancel the operation and return to the start of the application or task.**

  Used to return the user to the home deck, main menu, or an intuitive card within the application. To allow the user to exit an application or form prematurely, use a delete shield to confirm the action.

- **Skip: Used to lead to similar data, such as the next news article or email message.**

  When Skip is used, it should always be the first softkey. Can be repeated as a link at the end of a page as well.

- **View: Used to select an item in a menu of similar data, such as a list of stock quotes or email messages.**

  Must provide additional detail. If Times Square scrolling is used to display the list data, pressing View redisplays the data.

- **Details: Used as a link to get details of an item, such as a news article that is summarized by a headline.**

  If information continues on another page, then use More instead of Details.

- **More: Used as a link at the end of a page of data to see the next page of related and similar data.**

- **Back: Not used for entry query cards or as a** `<do type="options">` **with a** `<prev/>` **task.**

  Back returns the user to the previous card in the history list. Do not define a Back softkey, because users tend to rely on that rather than on the dedicated Back key. Back on the primary softkey should be assigned to the `<do type="accept">` task only if no label is defined for the `<do type="options">`. If Back is assigned to the `<do type="options">` label and there is no `<do type="accept">` task assigned, then two Back softkeys are displayed Therefore, if Back is required on the softkey, make sure that the card has a `<do type="accept">` task assigned to something other than `<prev/>`.

Conflicting Labels and Links

The following labels may conflict with items on a browser menu and may mislead the user.

- **Exit: May imply exiting the browser.**
- **Next: Often misinterpreted by users when used as a link.**
- **Home: May imply the browser's home card.**
- **Bookmark: May conflict with the browser menu.**
- **Menu: Automatically generated by the browser when multiple** `<do type="options">` **are defined.**

Identifying the Browser

A

Several classes of web clients could potentially access your site, but for the sake of simplicity, this appendix addresses four possible situations:

- A client that expects HTML
- A device with the UP.Browser 3.1 or 4.x that supports WML 1.
- A device with the Openwave Mobile Browser 5.0 or 5.x that supports WML 1.3
- All other WML browsers

To identify which client is accessing your site, investigate two different HTTP headers, `HTTP_ACCEPT` and `HTTP_USER_AGENT`. Although neither of these is part of the WAP specifications, they are both standard HTTP headers defined in RFC1945 (see `http://www.rfc-editor.org/rfcsearch.html`).

The first step is to parse the `HTTP_ACCEPT` header for the inclusion of `text/vnd.wap.wml`:

■ **Perl**

```
#!/usr/local/bin/perl

$acc = $ENV{"HTTP_ACCEPT"};
$ua = $ENV{"HTTP_USER_AGENT"};
if ($acc =~ "wml"){
    deliver wml
}
else{
    print' Location: http://mysite.com/index.html'."\n\n";
}
```

- **Java**

```
public void doGet (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException{
        String acc = req.getHeader("Accept");
        String ua = req.getHeader("User-Agent");
        ServletOutputStream out = res.getOutputStream();

        if (acc.indexOf("wml") != -1){
            deliver wml
        }
        else {
            res.setHeader(res.SC_MOVED_TEMPORARILY);
            res.setHeader("Location", "http://mysite.com/index.html");
        }
```

- **ASP**

```
<%response.buffer="true"
    Dim accstring
    Dim uastring
    uastring = request.ServerVariables("HTTP_USER_AGENT")
    accstring = request.ServerVariables("HTTP_ACCEPT")
If (InStr(accstring,"wml")) Then
    Deliver wml
Else
    Response.Redirect("/index.html")
```

In all of these cases, it has been established that the client sends WML in the `HTTP_ACCEPT` header and can therefore assume that WML should be delivered. If WML is not found in the `HTTP_ACCEPT` list, HTML is delivered. This ensures that HTML is delivered to web browsers and to spiders (crawlers, site indexers).

Once a WML device is found, it is possible to discriminate further to see exactly which device is accessing the site. The following code first looks for an Openwave graphical Mobile Browser, then for the Openwave text-based browser, and if that is not found, delivers content formatted for the generic browser.

- **Perl**

```
if($ua =~"UP.B" || $ua =~"UP/"){
    if ($ua =~ "GUI"){
        print "Location: /opwvgui/index.wml \n\n";
    }
else{
    print "Location: /opwv/index.wml \n\n";
    }
    else{
        print "Location: /generic/index.wml \n\n";
    };
```

- **Java**

```
if ((ua.indexOf("UP.B") != -1)) || (ua.indexOf("UP/") != -1)){
    res.setHeader(res.SC_MOVED_TEMPORARILY);
    if ((ua.indexOf("GUI")) != -1)){
        res.setHeader("Location", "/opwvgui/index.wml");}
    else{
        res.setHeader("Location", "/opwv/index.wml");
}
else{
    res.setHeader(res.SC_MOVED_TEMPORARILY);
    res.setHeader("Location", "/generic/index.wml");

};
```

- **ASP**

```
If((InStr(uastring, "UP.B")) || (InStr(uastring, "UP/")))
    If((InStr(uastring, "GUI"))
        Response.Redirect("/opwvgui/index.wml")
    Else
        Response.Redirect("/opwv/index.wml")
Else
    Response.Redirect("/generic/index.wml")
```

Summary of Graphical Mobile Browser Elements and Attributes

B

The following table lists the new UI WML+ features of the graphical Mobile Browser, along with some tips on how to use these features to make your applications more usable.

Element/Attribute	Tips	Examples
The <prev> element	Never define a <do type="options"> with a <prev/> task.	Every phone already has a hard-coded key to go back in the history. A Back on the softkey is redundant and is displayed on both softkeys if no other options are listed. This occurs when the user scrolls above the browser menu or below the last item.
General usability	Break information into screen-size segments of related information (wizard paradigm).	For making a purchase, create a card that allows the user to select an item, one for entering billing information and another for the shipping information.
	Combine element types for dates.	Depending on the application, use a combination of pop-up menus and input fields to allow users to enter the date. This helps the user know the required format. For example, use a pop-up menu for the month and an input field for the day.
	Define a title attribute for each <option> element.	The title attribute provides the ability to assign an intuitive or informative label (such as OK, Find, or Buy) to be rendered on the softkey.
Line breaks	Use line breaks between text and a pop-up or entry field when the field/pop-up sequence is too short to fit on the same line.	When the user is asked to pick an item from a list, make sure that the items in the list fit on one line.
Titles	Limit titles to 15 characters, when possible. Longer titles use more lines on the display, making it difficult for users to view the content on the card.	If a sequence of cards is used for looking up bank account information, use a consistent title throughout the set of cards, followed the description of the card.
	If more text is required, have the first 15 characters describe the sequence of cards, followed by text that describes the card.	For example, use <Card sequence>: <description of card task>.

Element/Attribute	Tips	Examples
Tables	Use tables to separate regions vertically.	Use in calendar applications or to separate data when displaying results.
	Use tables to align columns of text.	Use to align a list of information, such as a name with a phone number extension.
Buttons	Place buttons at the end of a card to complete an action and take the user to another URL.	Provide a button at the end of a form to allow users to submit the data.
	Use buttons in the beginning or middle of a card only to allow users to fill in or select data to be used in the originating card. Do not use icons on buttons.	Use a button that takes users to another card, which allows them to select information—for example, email addresses or a date. Use the selected information to be inserted into the originating card.
Test boxes	Do not start a card with an input field followed by a select list.	When the user tries the accelerator short cut, the input field is automatically activated.
	Use growing text boxes instead of fixed-line boxes if the information is likely to exceed one line and needs to be viewable by the user.	
	Use growing text boxes if the size or amount of data to be entered is not fixed.	This ensures that the user does not have to scroll through empty regions of a text box.
Radio buttons	Use in forms to replace selection list items.	This allows users to make a selection without having to go to the next URL. Use when there are only 3 or 4 items to select from.
Multiple-selection lists	Use to allow users to select multiple items.	This helps avoid user input and prevents having to parse typed information.
	Use to turn a setting on or off in a form.	Use a multiple-selection list to allow users to turn on an alert or default setting.
Pop-up menus	Use in forms instead of a selection list.	Using a selection list takes the user to another URL. Pop-up menus can be used instead of radio buttons when lists are more than 4 items long.
	Limit the number of items in the menu to 9 or fewer.	This makes it easier to scroll the list, particularly on phones with smaller displays.
Horizontal rules	Use to add a visual break between content regions.	Use to divide recipient and subject information from the text of an email message.

C Sample Application

This appendix presents a sample application with a set of forms that illustrate the guidelines described in this document and show how to modify applications for ease of use. The sample application has been designed to work well with the graphical Mobile Browser. The application was designed based on the following methodology:

- Determining who the user is
- Determining the user's goal
- Making it easy to accomplish the goal
- Making the application easy to navigate
- Scaling the application to perform only the necessary functions
- Creating consistency throughout the application
- Avoiding or reducing the amount of required text entry
- Avoiding situations that cause unnecessary errors

The WML code for these applications can be accessed online at

```
http://demo.openwave.com/styleguide/v5gui/index.wml
```

The entire directory structure is downloadable from

```
http://demo.openwave.com/styleguide/v5gui/code.zip
```

Travel Service

This example applies the steps for creating usable applications to the building of a search application that uses a sequences of forms, including the graphical Mobile Browser elements (pop-up menus, buttons, radio buttons, a variety of entry fields, and pop-up softkey menus) and selection lists, and screens displaying long text sequences.

- **Define the user.**

  The expected user of this application ranges from frequent users to those who may check every couple of weeks or once a month. The frequent user travels for business (30% or more) and may be familiar with flight schedules, airport code, and planes. The less frequent user is one who enjoys traveling in general, but most of the travel is for personal reasons.

- **Determine the goal.**

  Goals include tracking flight status (on time or delayed), seeking flight times, and making a reservation. The less frequent users include those who want to watch fares on a given route or are looking for specials.

- **Make the goal easily attainable.**

  Most travelers have a frequent flyer number, so the ability to search booked flights by the frequent flyer number is useful (though this is not implemented in the Travel sample application). For example, if a meeting finishes earlier or later than expected, this user wants to be able to look up a flight and make a reservation. For the less frequent traveler, who is interested in checking fares, the application should offer the ability to establish a list of fares to watch, as well as specials. Allowing users to select items from the accelerator key shortcuts lets them access the desired path quickly.

- **Make the application easy to navigate.**

  Frequent travelers want to get reservation information quickly, so when users log in for the first time, item allowing them to view their itinerary should be available. Additionally, a menu item for checking the flight status needs to be near the top of the menu so that travelers can find out their flight status quickly. Users should be able to check the flight status from their account information as well. Less frequent travelers want to be able to quickly check for special deals. Placing important and user-relevant information at the top of the menu makes it highly visible and easy to retrieve.Limit the application to the necessary functionality.

- **The highest priority tasks should be listed first.**

  The primary focus for this application is to provide quick and easy access to flight status information. Secondary functions, such as making a reservation, updating account information, and looking up how many frequent flyer miles have been accrued may require additional steps. Additional functions, such as setting alerts for changes in gates, times, and best fares, may also be accessible through secondary paths.

- **Make the application consistent throughout.**

  Using a consistent set of labels for elements or softkeys reduces the time the user needs to access option lists or to review softkey labels. For example, using the same labels for menu items or links from the flight status screen to make a reservation gives users a consistent way to complete the task.

- **Avoid text entry.**

  If the application remembers the user’s frequent flyer number, the user can access itineraries and the best fares list without having to enter data.The application should store the user’s personal information and preferences for ticket purchase. For example, when supplying the name on the credit card, the application fills in the user’s name and other information such as seat selection. When searching for a flight, the application should allow the user to search by flight number or airport code, as well as partial match for cities.

- **Prevent unnecessary user errors.**

  For users who do not know their flight number or airport code, it is helpful to be able to use a partial match to search for cities. The input field for such a search can be limited in length so that the user does not have to enter long text strings. Using pop-up menus for dates can prevent entry problems, and a calendar makes it easy for the user to select the correct dates. Cards that allow the user to confirm actions prevent an accidental purchase or reservation loss.

Application Overview

Figure C-1 shows how to design the application to take advantage of the graphical Mobile Browser user interface features. Note that the display size is not based on any particular phone; rather, the figures illustrate content and forward navigation. Backward navigation is discussed in the design summary section. In these examples, the `<do type="accept">` label on the Openwave Mobile Browser, WAP Edition, is displayed on the left softkey and `<do type="options">` labels are on the right softkey. Remember that this application is only an example, designed for a particular cultural environment. The actual fields and text should be modified for the appropriate market or culture.

Figure C-1. Travel Application

Design Summary

The design limits the number of steps the user must take to access the desired information while retaining as much intuitiveness and usability as possible. Although only one path is shown, this summary also discusses additional capabilities to meet the design goals.

General Navigation

The application allows the user to search by itinerary or flight status, make a reservation, establish a watch list of best fares for specified destinations, and view the current specials. It also allows users to set notifications to be informed when a new special is available from specified departure cities, see what planes are available on a given flight, and establish a profile of personal data.

The application design minimizes the number of steps that users must take to access the primary information that is relevant to their goals. For example, when the frequent user first enters the flight service screen, he or she can check on the status of a flight and make a reservation. The application stores the login information in a cookie, so that the next time the user accesses the application's home card, dynamic menu items appear. In this

case the first item changes from Login to Itineraries. This way the accelerator numbers to the other menu items do not change, and it also provides a quick way for users to view the status and other information of existing itineraries. Additional items such as “Fare watch” and “Account information” also become available. When accessing the “Account information,” the user can view, edit, and change profile information and establish notifications and alerts. The “Fare watch” item allows the user to create a list that displays the lowest cost flights from one city to another. For the less frequent traveler, a menu item for eSpecials presents the weekly specials.

When the user selects “Flight status,” there is an option to allow the user to select “By flight number” or to select by city or airport code. This means that users who do not know their flight number can check by other means. A partial city match, up to a limit of 8 characters, provides enough data to perform a search for the city, but more than the 3 characters required for an airport code. Once the user selects the flight number or airport code and enters the relevant date, he or she can check whether there are any delays. While viewing the flight status, the user can also check the seat availability and/or book that flight. The frequent traveler is able to quickly make a reservation using his or her frequent flyer information.

A second path from the main menu also allows the user to reserve a flight. This path differs from “Flight status” because it requires the user to enter more information, specifying airline, dates, locations, number of passengers, and class of service. Once the information is entered, the user can select the flight. Throughout this form, the user has the choice of continuing on to the next card in the sequence or going back to change the information. The application first allows the user to select the flight, then provides more details about the flight, such as a link to the type of aircraft and on-time record. Once the user confirms the departure and return flights, the application provides a card reviewing the reservation information and allowing the user to continue on to purchase the ticket, cancel out of the application, or change the reservation. When choosing to purchase the ticket, the user is presented with his or her contact and credit card information. If the user has logged in, the data is automatically preloaded into the appropriate fields and the additional preferences in the account are used. If not, the user can enter the information directly. Once again, the user can exit the application or continue with the purchase. Once the purchase is made, the application displays the confirmation number.

Backward Navigation

As stated earlier, the user can change the information while making a reservation. If the user chooses “Change reservation,” the first card in the series is presented and the cards between that card and the first card in the form are removed from the history. However, the data already entered is retained, so that the user can change just the specified field. This means that the user can easily traverse through the application again. To establish this ability, a context is used so that the cards along the path are removed. This way, if the user presses the Back key after cancelling out of the application, he or she does not have to navigate backward through the form. To change only the information on the previous card, the user can press the Back key instead of Cancel.

Developing for the Individual Browsers

It is important to remember that this application has been designed for the graphical Mobile Browser and may have usability constraints on other browsers. Therefore, develop or maintain the application for other nongraphical browsers.

Technical Summary

You can access this application at

```
http://demo.openwave.com/styleguide/v5gui/index.wml
```

This application is designed to be driven by real-time, dynamic data that is available from an airline reservation service. To provide dynamic data in this application, a WMLScript file (`times.wmls`) is used. The WMLScript randomly generates flight numbers and departure times, which are used later in the application.

There are several important GUI features of the application that are used repeatedly through different paths. Additionally, the application navigation and caching model is important because the user may want to back out of the search results or change a selected itinerary at any point in the application.

In the Reservations path, a form is presented to the user to select a departure and arrival. Both of these values are required, so the `emptyok="false"` attribute is used for both of these input elements. This ensures that the user cannot submit the form without entering values. The validity of these values must be checked by the server side of this application (which is not fully implemented).

The list of airlines is presented as a pop-up list, so that it occupies only a single line on the display. Selecting one of the options in the pop-up list assigns the value to the `name` attribute of the `select` element:

```
<select type="popup" name="airline" title="Pick Airline">
    <option value="Any">Any</option>
    <option value="AA">American</option>
    <option value="AW">America West</option>
    <option value="AS">Alaska</option>
    <option value="CO">Continental</option>
    <option value="DL">Delta</option>
    <option value="NW">Northwest</option>
    <option value="SW">Southwest</option>
    <option value="TW">TWA</option>
    <option value="UA">United</option>
    <option value="US">USAir</option>
</select>
```

Immediately after the airline pop-up list, the user is presented with the option to select departure and return dates. The calendar icon is active, and takes the user to a card that presents a calendar. This icon is made active by defining it as part of a <do type="button">.

```
Depart date:
    <do type="button">
        <go href="calendar2.wml">
            <setvar name="date" value=""/>
        </go>
        <img localsrc="calendar" src="" alt=""/>
    </do><br/>
```

A button can be presented with either an image as a label or with a text label. To use a text label, simply define the label attribute for the <do> element: <do type="button" label="Find flights">. All other properties, attributes, and acceptable elements in a <do> element apply to the definition of the button. If the user does activate the calendar icon, he or she is presented with a month view calendar. Each day in the calendar represents an <anchor> that is in a cell of its own. Selecting a date results in that value being filled in on the form.

Alternatively, the user can choose the month, day, and year of departure by using the <select> and <input> elements. Notice that all three of these elements are rendered on the same line of the display. This is possible because as long as there is room, the browser presents multiple GUI elements on the same line.

```
<select type="popup" name="dmonth" value="Mar">
    <option value="Jan">Jan</option>
    <option value="Feb">Feb</option>
    <option value="Mar">Mar</option>
    <option value="Apr">Apr</option>
    <option value="May">May</option>
    <option value="Jun">Jun</option>
    <option value="Jul">Jul</option>
    <option value="Aug">Aug</option>
    <option value="Sep">Sep</option>
    <option value="Oct">Oct</option>
    <option value="Nov">Nov</option>
    <option value="Dec">Dec</option>
</select>
<input name="dday" size="2" format="NN" value="08"/>
<select type="popup" name="dyear" value="2001">
    <option value="2001">2001</option>
    <option value="2002">2002</option>
</select><br/>
```

If you want elements to be rendered on a line of their own, you must use a
 between each element. Also, depending on the size and resolution of a device, elements may or may not appear on the same line.

The form is set up so that the user can either select a return date or leave it blank. If the user leaves the return date blank, the application performs a one-way flight search. If a return date is filled in, the user is presented with both outbound and returning flight choices. The flight numbers and departure times are generated by the WMLScript, but in a real application they would come out of a live database.

Once the user selects a flight or flights, he or she is presented with a card displaying all of the details, including the price of the tickets. If the user chooses to purchase the flight, another form is presented asking for credit card information, including the name on the card. The expiration date is rendered as a combined pair of widgets (a pop-up select list and an input element):

```
Exp. Date (Month/YYYY)<br/>
<select type="popup" name="month">
    <option value="Jan">Jan</option>
    <option value="Feb">Feb</option>
    <option value="Mar">Mar</option>
    <option value="Apr">Apr</option>
    <option value="May">May</option>
    <option value="Jun">Jun</option>
    <option value="Jul">Jul</option>
    <option value="Aug">Aug</option>
    <option value="Sep">Sep</option>
    <option value="Oct">Oct</option>
    <option value="Nov">Nov</option>
    <option value="Dec">Dec</option>
</select>
<input size="4" name="year" value="20" format="NNNN"/><br/>
```

Note that there is a line break `<br/>` between the descriptive label for the widgets and the widgets themselves. The `value` attribute of the `input` element prefills the `"20"` in the input field to reduce the number of keystrokes for the user, and the NNNN format string ensures that the user can enter only 2 more digits.

Aside from the GUI elements, the travel application also makes use of nested contexts within the application to ensure that users do not revisit the purchasing screens after they have completed a transaction. They can also return to the point where they can request a flight with a single keystroke. The application also employs a cancel shield so that the user does not inadvertently leave the application and lose any entered data.

An example of the context structure is that when the user chooses a particular flight, the following action is invoked:

```
<spawn href="reserveflto.wml">
    <setvar name="dpickflt" value="$dpickflt"/>
    <setvar name="dmonth" value="$dmonth"/>
    <setvar name="dday" value="$dday"/>
    <setvar name="dyear" value="$dyear"/>
    <setvar name="airline" value="$airline"/>
    <setvar name="class" value="$class"/>
    <setvar name="passnum" value="$passnum"/>
    <catch name="prev"/>
    <catch name="it"/>
    <catch name="bail">
        <onevent type="onthrow">
            <prev/>
        </onevent>
    </catch>
</spawn>
```

The <spawn> element directs the browser to the reserve flight deck and passes in the variables needed in the subcontext that is created via the <setvar> elements. Three different catch statements are included as part of this <spawn> element. The <catch name="prev"/> allows the user to back up directly to this card, and the <catch name="it"/> catches a throw event later in the application that allows the user to choose a new flight from the original list of flights. Finally, the <catch name="bail"> handles the situation where the user wants to restart the flight search. When this deck receives a throw event named "bail", the browser backs up one card in the history stack from the list of flights to allow the user to modify the flight search criteria and start over. The "bail" exception is thrown on the “Reservations” and “Ticket Purchase” screens if the user chooses to cancel. On the “Ticket Purchase” screen, the "bail" exception is thrown only after the user visits the cancel shield and reconfirms that he or she wants to leave the application.