CHAPTER 10

The Handheld Device Markup Language

ALTHOUGH THE WIRELESS APPLICATION PROTOCOL (WAP) is poised to be the world standard for screen phone content, the Handheld Device Markup Language (HDML) remains important in North America, and I recommend you gain at least some familiarity with it.

In this chapter, I introduce HDML and show you how you an use it to mark up your content for today's North American screen phones and wireless terminals.

Introducing HDML

HDML was the first device-specific markup language available for screen phones. Created by Phone.com, it has been widely licensed to handset manufacturers. Major wireless providers, including AT&T, offer mobile data services that carry HDML, and most North American wireless data handsets in use offer HDML browsers.

Unlike the other wireless technologies discussed in this book, the HDML standard is controlled by a single company. While the standard itself is open—anyone can develop HDML content—the direction HDML will take in the future is controlled by Phone.com.

HDML depends on the UP.Link Server, which provides server-side assistance for HDML browsers. The UP.Link Server gates HDML content from the Web to wireless terminals (see Figure 10-1 for a deployment view). As with other server-assisted wireless browsers, the UP.Link Server bridges the gap between media-rich Web content and constrained access devices. It provides network-specific services to the wireless network, converting the wireless protocols to Web protocols and making requests of the servers where the content originates—on the Web or on private enterprise networks.

Fortunately, it's not necessary to know much about the UP.Link Server to develop HDML content. Network providers make a UP.Link Server available and support it for their subscribers.

Developers who set out to learn HDML will find experience with other markup languages helpful. Understanding the Wireless Markup Language (WML) makes HDML easier to learn at a conceptual level, but knowing the HyperText Markup Language (HTML) well makes HDML easier to write. HTML and HDML tags are written using the same syntax, and some tags are the same in both markup

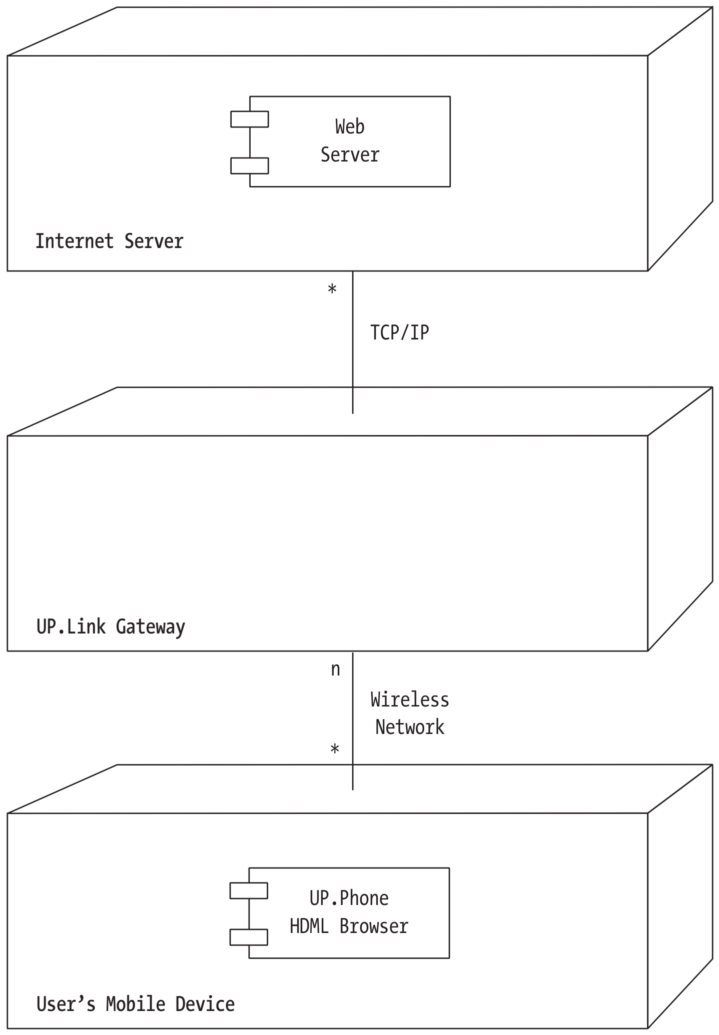


Figure 10-1. The deployment model for HDML services

languages. However, the organization of an HDML document is most similar to that of a WML document, and several important features of HDML, including tasks and variables, resemble those of WML.

This dichotomy originates in the history of HDML. Originally developed by Phone.com's previous incarnation, Unwired Planet, HDML was intended as an alternative to HTML for mobile devices on which HTML processing would be too expensive to be practical. As the adoption of HDML accelerated, a number of handset manufacturers interested in establishing an open standard for wireless data worked with Phone.com to create the WAP Forum, the industry association that developed and maintains the WAP standards. Thus, HDML has a syntax based on HTML, but includes concepts refined in WAP.

HDML or WML?

The existence of two complementary standards in a growing market leads inevitably to the question, "Which one should I use?" While there may be special cases, the basic answer in this case is simple: support both.

The differences between WML and HDML are rooted in syntax, not semantics. For conventional providers developing simple wireless applications that consist largely of information, the majority of the work involved in supporting both formats lies in selecting the appropriate tags for the language at hand. The limitations of WML and HDML devices are similar, and the two languages require similar formats and interface conventions. Content obtained from databases or other sources can be combined with HDML templates as easily as with WML templates, so there is rarely a good reason not to offer dual-platform information services.

The question becomes more difficult, however, when true interactive wireless applications are considered. These applications do more than display content; they interact with the user through client-side scripts that make them appear to be applications in their own right. Both HDML and WML provide the ability to create wireless applications, but their syntax and capabilities rule out the creation of cross-platform applications. While the organization and design of an application may be applicable to both platforms, the markup tags themselves are different, so you'll need to approach the content differently.

If you are looking to create a dynamic Web-based application, such as an e-mail client or a personal information manager, it's important to consider your target market as well as the technical differences between HDML and WML.

Wireless Web applications are much more sensitive to the market than are traditional applications. Their adoption is influenced by relationships with service providers, pricing, advertising, and a host of factors that have a much smaller impact on traditional software projects.

For example, in many cases, vertical markets already have established relationships with both kinds of vendor. For a wireless application to be useful, it must run on the end customer's selected hardware over the provider's network. You won't win points from a customer for delivering a custom WML application for an HDML-compliant Alcatel wireless handset.

Similarly, if you are targeting an application for a particular service's customers, bear in mind the limitations of that service. For example, as of this writing, AT&T's Pocket.Net service is based on HDML, so applications targeted at Pocket.Net subscribers should certainly be HDML-based.

The features of WML and HDML are technically similar. Both languages support variables, and both provide limited on-device processing of data. The WAP stack provides WMLScript, a scripting language that provides conditional evaluation and other primitives that you could only get from a server in an HDML application. However, HDML has a few other features, such as activities and iconic soft-key labels (more information about both of these appears later in this chapter), that are presently unavailable to WAP developers.

Your First HDML Page

The basic syntax of HDML is simple enough that it will become clear from a single example:

```
<HDML VERSION=3.0>
    <- A simple HDML deck ->
    <DISPLAY NAME=hello>
    Hello world!
    <BR>
    How are you today?
    </DISPLAY>
</HDML>
```

Note the strong similarities to HTML: tags are contained within angle brackets < and >. Tags that accept arguments, such as the `<DISPLAY>` tag, are written using the same angle brackets as empty tags such as `<BR>`. Comments can be included between the delimiters `<--` and `-->`.

As in HTML, many tags have attributes that further describe their behavior. Attributes are specified as named values within the angle brackets but after the tag's name, such as the `NAME` attribute of the `<DISPLAY>` tag in the example. (These attributes are described in detail in the section "HDML for Web Developers," later in this chapter.)

You must begin any HDML document with the `<HDML>` tag. Because only one document can occupy a file, every HDML file begins and ends with this tag.

HDML documents are organized as decks of cards, just as WAP documents are. Thus, the `<HDML>` tag delimits a deck. The cards within the deck can specify actions, declare or evaluate variables, and display content. (A card may do any or all of these things, although as you'll see, one card can't prompt for both text and selection input.) Any of the tags `<DISPLAY>`, `<NODISPLAY>`, `<CHOICE>`, and `<INPUT>` can be used to specify a card.

Another feature of HDML that differs from HTML is the use of variables, which play an important role in handling user input in HDML. Variables are set using the `<ACTION>` tag and can be evaluated using the `$` operator. The best way to explain their use is through an example:

```
<HDML VERSION="3.0">
  <NODISPLAY>
    <ACTION TYPE=ACCEPT TASK=GO DEST=#main
     VARS=pi=3.14159&e=2.71828>
  </NODISPLAY>
  <DISPLAY NAME=main>
  PI is $pi.<BR>
  e is $e.
  </DISPLAY>
<HDML>
```

This card uses a hidden card—declared with the `<NODISPLAY>` tag—to set the variables `pi` and `e`. The second card of the deck evaluates these variables in the content that is displayed using the `$` operator.

Browsers, Tools, and SDKs

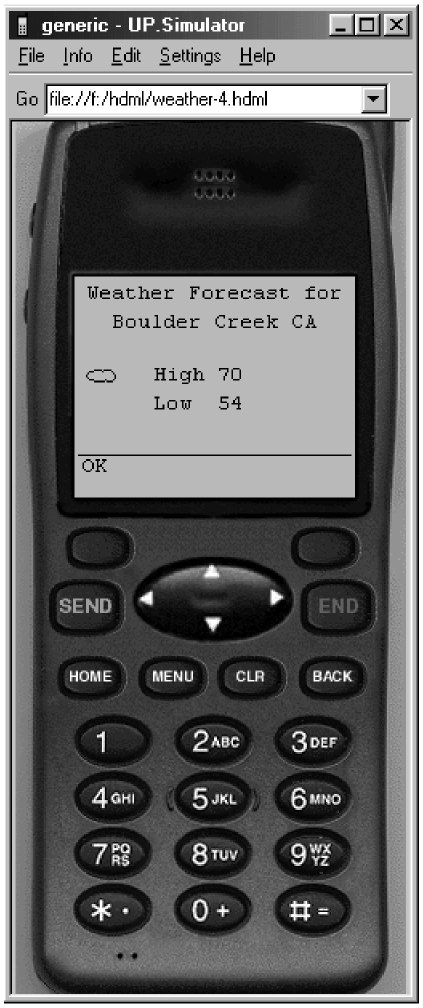
A variety of handset manufacturers, including Motorola, Alcatel, Mitsubishi, and Samsung, now offer phones with HDML browsers. At present, there are no HDML browsers for laptops, Personal Digital Assistants (PDAs), or hybrid phone/computing devices.

Despite its well-established presence in the industry, no proven tools exist that enable What-You-See-Is-What-You-Get (WYSIWYG) editing or validation of HDML; for the present, therefore, HDML content must be developed with a text editor. Any simple text editor will do, including WordPad, BBEdit, or emacs. Simply compose using standard ASCII characters, and save the result as a flat text file with the extension ".hdml". (When saving HDML files from a Windows text editor, don't forget to specify the filename in quotes, or your file will end up with the suffix ".hdml.txt").

To preview HDML content, you need a copy of the Phone.com UP.SDK software developer kit (SDK). It includes a simulator capable of displaying HDML files from remote servers or a local file system. Figure 10-2 shows a screen shot of the simulator in action.

HDML for Web Developers

If you have used another markup language, you'll find it easy to learn HDML. HDML draws on common concepts found in both WML and HTML, although it has some new ideas of its own as well.


generic - UP.Simulator
File Info Edit Settings Help
Go file://f:/hdml/weather-4.hdml
SEND
END
HOME
MENU
CLR
BACK

```
Weather Forecast for
  Boulder Creek CA

     High 70
     Low  54

OK
```

Figure 10-2. The Phone.com UP.SDK HDML browser

Cards and Decks

As with WML (see Chapter 8), the fundamental organizational metaphor behind HDML is the deck of cards. *Cards* act as containers for all other objects; they describe actions, display content, and control interactions with users. There are also hidden cards, which encapsulate tasks or manage variables. Cards are grouped together into *decks*. Usually, the cards in a deck relate to a common purpose or subject, such as a stock quote, news headline, or weather report.

Attributes of Cards and Decks

Many HDML tags accept *attributes*, which control how the client renders the tag. The attributes of cards and decks are paramount, because they determine such aspects of your content as who can access your decks and how navigation between decks and cards behaves. In the following sections, I discuss the attributes of the cards and decks shown in Table 10-1.

TAG	**ATTRIBUTE**	**PURPOSE**
`<DISPLAY>*`	`BOOKMARK`	Specifies the URL that should be saved when a card is bookmarked.
	`NAME`	Specifies a card's name for navigation purposes.
	`MARKABLE`	Indicates whether a card can be bookmarked (true) or not (false).
	`TITLE`	Specifies a card's bookmark title.
`<HDML>`	`VERSION`	Specifies the version number of HDML being used.
	`ACCESSPATH`	Specifies the path of decks that are allowed to link to an access-controlled deck.
	`ACCESSDOMAIN`	Specifies the domain of decks that are allowed to link to an access-controlled deck.
	`PUBLIC`	Indicates whether access to a deck is restricted (false) or not (true).
	`TTL`	Specifies a deck's time to live, in seconds.

*The attributes of the `<DISPLAY>` tag are also attributes of the `<CHOICE>` and `<INPUT>` tags.

Table 10-1: HDML Deck and Card Attributes

Deck Attributes

The attributes of a deck determine the behavior of all the cards in it: whether they can be bookmarked or not (by default), how long they can remain in the browser's cache, and whether the deck can be linked to other decks.

The `<HDML>` tag, mentioned previously, introduces each deck. Its `MARKABLE` attribute specifies whether or not cards within the deck can, by default, be bookmarked. This flag may be overridden for specific cards.

The `PUBLIC` attribute indicates whether or not a deck's contents can be accessed from other decks. By default, any deck can be linked to any other deck. You can restrict access to a deck by specifying a value of `FALSE` for the `<HDML>` `PUBLIC` attribute.

When PUBLIC is set to FALSE for a given deck, other decks can only be linked to it if they have the base URL specified in its ACCESSDOMAIN and ACCESSPATH attributes. The default for the PUBLIC attribute is FALSE. Note that a deck with restricted access cannot have bookmarkable contents, because the bookmark would access the deck from outside the specified access domain and path.

The final attribute of the <HDML> tag is the optional TTL attribute, which specifies the amount of time a deck may remain cached. (TTL stands for "time to live" and is given in seconds.) Rapidly changing data, such as stock quotes, call for a low TTL, while content that does not change can last longer. The following example indicates that the deck should be cached for twelve hours:

```
<HDML VERSION=3.0 TTL=43200>
<DISPLAY NAME="result" MARKABLE=FALSE>
  <-- Card's content -->
</DISPLAY>
</HDML>
```

By default, content will remain cached for no more than thirty days. Even without a TTL attribute, however, data may end up cached for considerably less time, as browsers may delete content viewed infrequently.

Card Attributes

A card's behavior is primarily determined by four attributes of the <DISPLAY> tag and the <NODISPLAY>, <CHOICE>, and <INPUT> card tags. The NAME attribute specifies a name for the card. Card names are used for navigation and are similar to anchor references within HTML text. Every card can be uniquely identified by the URL of the deck, followed by the hash symbol (#), followed by the name of the card; for example, http://hdml.apress.com/example.hdml#hello. As you'll see in the next section, it's also possible to navigate among cards using relative URLs.

HDML provides control over the behavior of bookmarks via the TITLE, MARKABLE, and BOOKMARK attributes of the <DISPLAY> tag. A card can only be bookmarked if its MARKABLE attribute is TRUE. (This attribute for a specific card overrides the default, which is determined by the MARKABLE attribute for the deck.) The TITLE attribute of a bookmarkable tag specifies a short title for the card within a bookmark; the URL that a bookmark opens can be set using the BOOKMARK attribute. By default, the current card's URL becomes the bookmark's URL.

Consider a deck that is used to obtain telephone numbers from a directory. You wouldn't want users bookmarking the results card or cards where form input has already taken place. In fact, you probably would want only the first card of the deck, where users enter the desired name, to be bookmarkable. Your HDML in this situation might look like this:

```
<HDML VERSION=3.0>
<DISPLAY NAME="home" TITLE="Moonlight Directory" MARKABLE=TRUE>
  <ACTION TYPE=ACCEPT TASK=GO DEST="#InputChoice">
    <-- Card content -->
</DISPLAY>
<CHOICE MARKABLE=FALSE NAME="InputChoice"
  KEY=choice IKEY=num IDEFAULT=1>
  <-- Card content -->
</CHOICE>
<ENTRY MARKABLE=FALSE NAME="Last" KEY=last FORMAT="*a">
  <-- Card content -->
</ENTRY>
<ENTRY MARKABLE=FALSE NAME="First" KEY=first FORMAT="*a">
  <-- Card content -->
</ENTRY>
</HDML>
```

Types of Cards

There are three broad categories of HDML cards:

- Cards that display information, such as formatted text or images.
- Hidden cards that specify variables or encapsulate actions available to the user, such as navigation to other pages.
- Cards that accept input from the user.

In most decks, these categories are mixed and matched to create interactive applications. For example, an application that searches an intranet phone directory will use cards with actions and input to enable users to look up coworkers' names and dial their phone numbers. The first card would be an input card in which the user enters a name. This card would link to a hidden card, which would wrap up the form content and post it to the server for a response. The response deck would contain a content card that displayed the phone number and included an associated action that dialed the number on the card.

Displaying Information

HDML decks can display plain text (there is no support for styles or different fonts) and images. Although it is minimalist in comparison to either WML or HTML, the text formats it supports are reasonable considering the capabilities of the target devices, and quite sufficient to display a wide variety of information. Table 10-2 summarizes the tags available for formatting display information on HDML cards.

TAG	**ATTRIBUTE**	**PURPOSE**
`<BR>`		Inserts a line break.
`<CENTER>`		Centers the current line.
`<IMG>`	`ALT`	Specifies a text label for an image.
	`ICON`	Specifies the source of an image found in ROM (by name).
	`SRC`	Specifies the source of an image obtained from the wireless Web (by URL).
`<LINE>`		Indicates that the current line should not be broken.
`<RIGHT>`		Right-justifies the text that follows.
`<TAB>`		Indicates that display should continue at the next tab stop.
`<WRAP>`		Indicates that a wrapping line follows (paragraph form).

Note: These are all empty tags.

Table 10-2: HDML Tags for Card Formatting

Text Presentation

HDML supports two kinds of text display: wrapped text (paragraphs) and a marquee scrolling text line. Wrapped text is appropriate for most purposes, including text paragraphs, news stories, and data display. The `<WRAP>` tag works much like the `<P>` tag in HTML; it goes at the beginning of each chunk of word-wrapped text.

Occasionally, you may need to display a line of text as an unbroken string, such as an e-mail header. The `<LINE>` tag comes in handy for this purpose:

```
<HDML VERSION=3.0>
<DISPLAY>
  <LINE>
```

```
The quick brown fox jumped over the lazy dog adroitly.
<WRAP>
After her followed four slow green turtles.
</DISPLAY>
</HDML>
```

The contents of the `<LINE>` tag will scroll repeatedly across the display so that the entire contents can be read on a single line. (Unfortunately, it's hard to demonstrate this feature in a paper book!) This tag should be used sparingly, as a host of moving lines is difficult to read.

Text Formatting

Formatting is controlled by a set of tags that specify a particlar operation for the browser, such as "center the following text," "place a tab," or "end a line." Unlike in HTML, these tags do not enclose the text being formatted; rather, they are placed at the begining of the run of text to which they apply, and the indicated formatting continues until the next formatting tag.

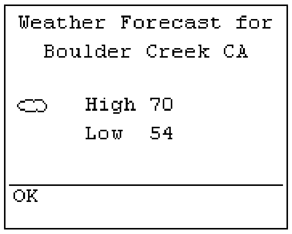
Line breaks can be inserted at any point using the empty `<BR>` tag. You can also use it in conjunction with the empty `<TAB>` tag, which inserts a tab, to create simple tables. By default, all text is left-justified. Text regions can be centered or right-justified using the empty `<CENTER>` and `<RIGHT>` tags.

The following example shows some of these text formatting features in action. The initial text is centered with the `<CENTER>` tag, while the weather report appears in a two-column table.

```
<DISPLAY NAME="result" MARKABLE=FALSE>
  <WRAP>
  <CENTER>
      Weather Forecast for<BR>
      Boulder Creek CA
  <LINE>
  <TAB>High<TAB>70<BR>
  <TAB>Low<TAB>54<BR>
</DISPLAY>
```

Displaying Images

Some HDML browsers can display simple monochrome images. The empty `<IMG>` tag specifies an image's text label and origin. Images may be obtained from origin servers on the Web or selected from a set of images the manufacturer stores in the smart phone's Read-Only Memory. This set of images is defined by Phone.com and consists of almost 200 clip-art images appropriate for wireless applications. These images are designed to look good on the phone, and may have been redrawn to best suit a particular phone's display. Whenever possible, select your images from the phone collection, as they will both load faster and look better on the display. The particular images available may depend on the version of the HDML browser you are using; check the Phone.com SDK documentation when selecting an image for your content.



```
Weather Forecast for
  Boulder Creek CA

     High 70
     Low  54

OK
```

Figure 10-3. A card with formatted text and an image

You specify the image you want using the `ICON` attribute if the image is in the phone, or the `SRC` attribute if it originates from a Web site. Using the `<IMG> ALT` attribute, you should also provide an alternate text title for each image. This attribute is optional in HDML, but in practice it is essential, as some users may not be able to view the images you include with your content.

Adding an icon from the handset's ROM to our weather page gives us an HDML file like the one that follows. (The resulting display is shown in Figure 10-3.) The cloud image is only one of several weather-related images available from the HDML browser.

```
<DISPLAY NAME="result" MARKABLE=FALSE>
  <WRAP>
  <CENTER>
      Weather Forecast for<BR>
      Boulder Creek CA
  <LINE>
  <IMG ICON="cloud" ALT="clouds"><TAB>High<TAB>70<BR>
  <TAB>Low<TAB>54<BR>
</DISPLAY>
```

If you need an icon that isn't available in the cache, you can always specify a URL source for the image, as shown here:

```
<DISPLAY NAME="result" MARKABLE=FALSE>
  <WRAP>
  <CENTER>
      Weather Forecast for<BR>
      Mercury Desert
  <LINE>
  <IMG SRC="scorched.gif" ALT="scorched"><TAB>High<TAB>680<BR>
  <TAB>Low<TAB>-354<BR>
</DISPLAY>
```

Displaying Special Characters

As in other markup languages, there are certain characters in HDML that can't be used except as parts of the markup tags themselves. HDML reserves the characters `<`, `>`, `".&`, and `$` to define tags and manipulate variables. If you need one of these characters elsewhere, you can insert it using one of the special empty tags shown in Table 10-3.

CHARACTER	TAG	PURPOSE
`<`	`&lt;`	"less than" symbol
`>`	`&gt;`	"greater than" symbol
`"`	`&quot;`	double quotation mark
`&`	`&amp;`	ampersand
`$`	`$dol`	dollar sign

Note: These are all empty tags.

Table 10-3: Reserved HDML Characters and Their Corresponding Entities

Tasks and Actions

`<ACTION>` tags assign tasks to be performed with specific interface elements. These tasks include navigating to other decks, returning to a previous deck, and making a phone call. Some interface elements on the handset itself—such as the soft keys, help key, and delete key—can be assigned to tasks. In general, these are keys, but they may be other controls, such as a rocker switch or touchscreen. (In the following discussion, I refer to them simply as *keys* for brevity.) The actual kind and nature of these buttons may change from manufacturer to manufacturer.

Actions may be defined either at the card level or at the deck level; if you assign an action at the deck level, it becomes the default for the entire deck, but may be overridden by a particular card. A card may contain one or more actions.

Assigning a Task

The empty tag `<ACTION>` binds an interface element to a task. Interface elements recognized by the browser include the following:

- The **ACCEPT** button.
- The **HELP** action (which may or may not be a physical control on the phone).

- The **PREV** action (which also may or may not be a physical control on the phone), used by the browser to navigate to the previously viewed deck.
- The soft keys, **SOFT1** and **SOFT2**. These keys have no physical labels; a label must be specified to appear on the liquid crystal display (LCD) in conjunction with the key.
- The **SEND** button, which initiates a voice call.
- The **DELETE** button.

Some of these elements are associated with expected phone-related behaviors, of course. For example, users expect the handset to dial a number when they press the SEND button, so assigning this button to the task of clearing a form, for example, is a bad idea. In general, it is best to assign most tasks to the ACCEPT buttons or to one of the soft keys.

The `<ACTION>` tag accepts the attributes listed in Table 10-4. I discuss each of these in the following sections.

ATTRIBUTE	**PURPOSE**
`<ACTION>`	Links a task to an interface element.
`DEST`	Specifies the destination of a GO or GOSUB task.
`ICON`	Specifies the name of an image from ROM to be used as a soft key label.
`LABEL`	Specifies a label for a soft key.
`METHOD`	Specifies whether a navigation should result in a GET or POST request.
`NUMBER`	Specifies a phone number to be called.
`REL`	When set to the value NEXT, "prefetches" the indicated DEST resource.
`SRC`	Specifies the source URL for an image to be used as a soft key label.
`TASK`	Specifies the task to be performed.
`TYPE`	Specifies the type of interface element to be assigned a task.

Table 10-4: Attributes of the `<ACTION>` *Tag*

Kinds of Tasks

The browser recognizes the following tasks:

- The GO task requests a URL.
- The PREV task displays the previous card to the user. (If no previous card is available, the PREV task is equivalent to the CANCEL task.)
- The CALL task initiates a voice call to a specified number.
- The NOOP task does nothing. It is used to hide the default behavior of an interface item or to override a deck's <ACTION> tag for a specific interface item.
- The GOSUB task pushes the current activity on the activity stack and requests a URL.
- The CANCEL task cancels the current activity and returns the user to the caller's CANCEL card.
- The RETURN task returns to the previous activity as specified by the caller's NEXT card.

The relationship between an interface element and a task is established through the <ACTION> TYPE and TASK attributes. TYPE specifies the interface element, and TASK, not surprisingly, indicates the task to be performed:

```
<ACTION TYPE=ACCEPT TASK=GO DEST="http://hdml.apress.com/index.hdml">
```

In this example, the ACCEPT key has been programmed to display the first card of the deck at http://hdml.apress.com/index.hdml when pressed.

Other attributes of the <ACTION> tag are used to control its behavior. The SRC and ICON attributes can specify an image resource for use as the label of a soft key's label, using the same syntax as the <IMG> tag.

The REL attribute can specify "prefetch" of the indicated resource. The following code

```
<ACTION TYPE=ACCEPT TASK=GO DEST="football.hdml" REL=NEXT>
```

causes the browser to fetch the deck at football.hdml while the current card is being viewed. Prefetching content increases the perceived speed at which a site functions—but it can increase airtime costs unnecessarily, as the user might not view some prefetched content (but must pay for the time spent fetching it anyway). The REL attribute should therefore be used carefully.

Data from HDML forms is sent to servers using the `GO` task in conjunction with the `METHOD` and `POSTDATA` attributes. By default, all navigation operations use the `HTTP GET request` method. Specifying `METHOD=POST` instructs the UP.Link server to send form data by using the `POST` method instead. (You can also specify the `GET` method using `METHOD=GET` to be explicit.) When the `POST` method is in use, the `POSTDATA` attribute should be set to an ampersand-delimited list of variables and names for the server to process, for example:

```
<ACTION TYPE=ACCEPT TASK=GO
  DEST="http://masamune.lothlorien.com/wx"
  METHOD=POST
  POSTDATA="choice=$menu&city=$city&state=$state&zip=$zip">
```

This sends the form variables named `choice`, `city`, `state`, and `zip` to the server, along with the values of the HDML variables `menu`, `city`, `state`, and `zip`.

Other attributes of the `<ACTION>` tag are used to specify arguments for a particular task. In the last example, we saw the `DEST` attribute used to specify the location of the deck to be displayed. The `DEST` attribute may be any partial or whole URL, evaluated relative to the current deck's URL. Similarly, the `NUMBER` attribute specifies a telephone number for the `CALL` task.

Although there is no way to set HDML variables directly, it is possible to set them from within an action using the `VARS` attribute. `VARS` accepts a list of HDML variables and their values in the same format as the one used by the `POSTDATA` attribute. For example, the following `<ACTION>` tag sets the default value for the variable `ZIP`:

```
<ACTION TYPE=ACCEPT TASK=GO DEST=#result VARS=ZIP=95006>
```

Actions Spanning an Entire Deck

An `<ACTION>` tag contained within a card assigns a task only for that card. There may be times when you'll want to replace the default action of an interface element over the scope of an entire deck. To do this, simply place the `<ACTION>` tag after the `<HDML>` tag that begins the deck. For example, you might want to offer help information that will be the same across most or all of the cards in a deck, as shown in this example:

```
<HDML VERSION=3.0>
  <ACTION TYPE=HELP TASK=GO DEST="#help">
... <-- Other deck cards here -->
<DISPLAY NAME="help">
 <ACTION TYPE=ACCEPT TASK=PREV>
 <WRAP>
 You can obtain the weather by entering
```

```
  either the ZIP code of the place of
  interest, or the city and state.
</DISPLAY>
</HDML>
```

From any card in this deck, activating the browser's HELP interface will display the card named `"help"`.

You may think that you can do the same to set variables across a deck using this method, but you can't. The `VARS` attribute of the `<ACTION>` tag isn't a task in its own right, but must be contained within another task. You can't simply embed it in any `<ACTION>` tag in the deck either, because the variables will only be set when the task is performed. Fortunately, there's another way to have the same effect.

The `<NODISPLAY>` tag creates a hidden card that immediately invokes its `ACCEPT` or `PREV` interface element. It can be used at the beginning of a deck to initialize variables that will be needed later or to encapsulate several references to a single URL. (I'll use this trick later in this chapter in "Integrating Actions and Input." The basic idea is to collect a form's various post operations on a single card, rather than scattering them across multiple cards. This makes making changes to a deck easier.)

The deck shown here demonstrates the difference between a hidden card that sets a variable and a deck `<ACTION>` tag:

```
<HDML VERSION=3.0>
<ACTION TYPE=HELP TASK=GO DEST="#help" VARS=setwhere=help>

<NODISPLAY>
  <ACTION TYPE=ACCEPT TASK=GO DEST="#home" VARS="setwhere=hidden">
</NODISPLAY>

<DISPLAY NAME="home" TITLE="Weather" MARKABLE=TRUE>
  <WRAP>
  The variable setwhere was set to $setwhere.
</DISPLAY>
<DISPLAY NAME="help">
 <ACTION TYPE=ACCEPT TASK=PREV>
 <WRAP>
  This card simply sets the setwhere variable.
</DISPLAY>
</HDML>
```

When this deck is first viewed, the display will read "The variable setwhere was set to hidden." But after a user has invoked HELP and then returned from the help card, the message will change to "The variable setwhere was set to help."

The <NODISPLAY> card, like any other card, occupies memory in the cache and takes the browser some time to process. Because of this, you should avoid creating paths that require the browser to navigate through more than one hidden card at a time, so that the user does not encounter a lengthy period of apparent inactivity.

Using Activities to Organize Decks

HDML activities are a mechanism for organizing decks. They provide some of the same structure as functions do in a structured programming language. An activity can be invoked, and when it is complete, control is returned to the calling activity. Control can be returned with either a RETURN task—if the activity has been successfully completed—or a CANCEL task—if the user has cancelled the activity in progress.

Within a single activity, all variables have global scope across cards and decks. Variables of the same name can be used in different activities without conflict.

Consider the process of reading e-mail. The user begins by perusing subjects, an activity unto itself. He or she then selects a particular message, initiating a new activity—reading the message. At this point, there are now two activities—one in progress (reading a message) and the other suspended (perusal of subjects). The user could then add a third activity to the queue by replying to the message. This activity can either be terminated by the user (if the reply is cancelled) or ended normally (if the message is sent). In the case of a termination, the user would return to the message display; if the message was sent, the user would return to the list of messages. Each of these activities can use a different set of variables and one or more decks. The activity stack allows a deck within a specific activity to refer to a calling activity (the preceding one) without previous knowledge of its behavior or location.

The GOSUB task invokes a new activity. Like the GO task, it navigates to the deck specified in its DEST URL. Unlike GO, however, it declares a new activity context, creating a new name space for all variables. Previous variables' names and values are saved, but cannot be accessed by the current deck or others until a RETURN or CANCEL task occurs. These variables may be shared with the previous activity using the RECEIVE and RETVALS attributes of tasks.

When a GOSUB task is specified, attributes of the <ACTION> tag are used to specify how variables are communicated between the two activities, and where the browser should go when the activity is completed.

The NEXT and CANCEL attributes of GOSUB specify which card should be shown *after* the DEST card that was specified for the GOSUB activity has been invoked and the activity completed. The RECEIVE attribute may be used to assign the variables used by the browser to contain results once the activity is completed.

Whether the activity ends with a CANCEL or a RETURN task, the variables specified by the RECEIVE attribute of the GOSUB task can be set using the value of RETVALS. Both RECEIVE and RETVALS should contain semicolon-delimited lists of variables to be set, but RECEIVE indicates the receiving variables used by the browser, while RETVALS contains the values to return. The first value in RETVALS is stored in the first variable in RECEIVE, the second value in the second variable, and so on.

Here's an example of two activities, which we'll call "home" and "sub":

```
<HDML VERSION=3.0>
<DISPLAY NAME="home">
  <ACTION TYPE=ACCEPT TASK=GOSUB DEST="#sub"
   NEXT="#next" CANCEL="#cancel"
   RECEIVE="result;">
  This is a specific activity, like reading message headers.
</DISPLAY>
<DISPLAY NAME="next">
  <ACTION TYPE=ACCEPT TASK=GO DEST="#home">
  You've returned from the nested activity.<BR>
  result is $result.
</DISPLAY>
<DISPLAY NAME="cancel">
  <ACTION TYPE=ACCEPT TASK=GO DEST="#home">
  You've cancelled the nested activity.<BR>
  result is $result.
</DISPLAY>
<DISPLAY NAME="sub">
  <ACTION TYPE=ACCEPT LABEL="Return" TASK=RETURN
   RETVALS="result;">
  <ACTION TYPE=SOFT2 LABEL="Cancel" TASK=CANCEL
   RETVALS="cancel;">
  This is a nested activity.
</DISPLAY>
</HDML>
```

Figure 10-4 shows the how the display screen will appear as the browser enters each of the activities and the variables are reset accordingly. When the deck is first opened, "home" is the current activity. Pressing the ACCEPT key creates a new activity, the "sub" activity, which will set a return value to either "result" or "cancel". The user can exit this activity by pressing either ACCEPT (which is assigned to the RETURN task) or SOFT2 (which has been assigned to the CANCEL task).

It's important to remember that HDML provides support for activities through the *activity stack*, a hidden part of the implementation of the HDML browser, and the navigation task GOSUB. This stack contains each activity's variable context, RECEIVE and RETVALS, NEXT URL, and related information. There are no explicit tags that specify an activity, its bounds, or how or where it can be invoked. Thus, using activities in HDML can be a little like object-oriented programming in a procedural language—you can do it, but the language doesn't offer any concrete mechanisms to help.

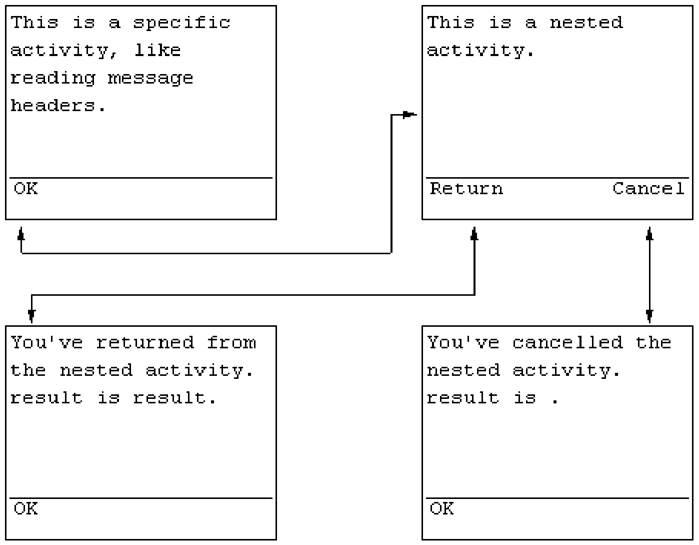

This is a specific activity, like reading message headers.
OK
This is a nested activity.
Return
Cancel
You've returned from the nested activity. result is result.
OK
You've cancelled the nested activity. result is .
OK

Figure 10-4. Flow between activities

The GOSUB task always starts a new activity, using system resources in the process. Be sure therefore that when a GOSUB task invokes a deck, it returns control to the calling deck with either a RETURN or a CANCEL task to end the new activity. Similarly, avoid using RETURN or CANCEL tasks in decks that can be accessed by GO, as they will end the current activity and may cause the browser to return to an activity other than one you intend.

Managing User Input

HDML supports both single-choice menu selections and text or numeric input. The <CHOICE> and <ENTRY> tags are used to present input choices and accept input from the user. Table 10-5 summarizes the tags and attributes for input handling in HDML.

TAG	ATTRIBUTE	PURPOSE
<CHOICE>		Indicates a choice card.
	DEFAULT	Specifies a default choice if the variable named by KEY is empty.
	KEY	Specifies the variable to contain the choice's VALUE attribute.

Table 10-6 *(continued)*

Table 10-6 (continued)

TAG	**ATTRIBUTE**	**PURPOSE**
	IDEFAULT	Specifies a default choice by index if the variable named by IKEY is empty.
	IKEY	Specifies what variable is to contain the choice's index.
	METHOD	Indicates whether the list should be numbered or unnumbered.
<CE>*		Indicates that this item is a choice that may be selected.
	VALUE	Specifies the value of a choice item.
<ENTRY>		Indicates an input card.
	DEFAULT	Specifies a default value if the variable named by KEY is empty.
	EMPTYOK	When true, allows empty inputs.
	FORMAT	Specifies what kind of input is valid for the entry card.
	KEY	Specifies a variable to contain the input.
	NOECHO	When TRUE, blocks input echo to the user.

*This is an empty tag.

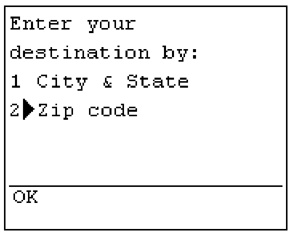
Table 10-5: HDML Tags for Card Input

The Choice Card

The <CHOICE> tag creates a card that presents a menu of choices like the one shown in Figure 10-5. The user navigates among the choices on the screen using the arrows or number keypad, then presses the OK button when the browser highlights the desired choice. Cards featuring this tag are called *choice cards*.

The attributes of the <DISPLAY> tag also apply to the <CHOICE> tag; it also has several more attributes that control the behavior of the choice interface.

Every <CHOICE> tag must contain at least one <CE> tag (which is short, one would guess, for "Choice Entry," but that's an unsolved mystery) to indicate choices to the user. The browser records the user's selection in a variable indicated by the <CHOICE> tag.


```
Enter your
destination by:
1 City & State
2▶Zip code

OK
```

Figure 10-5. A choice card

```
<CHOICE NAME="InputChoice"
  KEY=choice IKEY=num DEFAULT=zip>
  Enter your destination by:
  <ACTION TYPE=ACCEPT TASK=GO DEST="#$num">
  <CE VALUE=city>City & State
  <CE VALUE=zip>Zip code
</CHOICE>
```

In the previous example, the `<CE>` tag declares two choices. The first is the value `city`, and the second is the value `zip`. The user's selection is returned in the HDML variable named `choice`, and the index of the choice (counting from 1) is returned in the HDML variable `num`. Thus, if the user selects the first choice, the value of the HDML variable `choice` will be `city`, and the value of `num` will be `1`. If he or she picks the second choice, `choice` will be `zip`, and `num` will be `2`.

The attribute `KEY` of the `<CHOICE>` tag names a variable that will contain the user's selection. The browser will assign this variable whatever value is contained in the `VALUE` attribute of the selected choice. Similarly, the `IKEY` attribute names the variable to contain the index of the selected item. The `KEY` attribute is best used to assign human-readable values to variables for display in the current deck (such as phone numbers, prices, and so on), while the `IKEY` attribute can be used to return menu choices to a server for processing.

You can present two kinds of choices in HDML: *numbered* and *unnumbered*. By default, choices are numbered, and the user can make a selection with the appropriate number key. If, however, you set the `METHOD <CHOICE>` attribute to `ALPHA`, the choices will be displayed without numbers, and the only way to select an item will be to scroll to it and press the ACCEPT key. While there might be stylistic reasons for wanting a menu of choices without numbers, don't be too quick to create one. Unnumbered choices require the user to scroll through a list and enter multiple keystrokes, and therefore, are much slower to use than numbered ones.

The `DEFAULT` and `IDEFAULT` attributes of the `<CHOICE>` tag are used to select an initial choice. If the variable assigned to the `KEY` or `IKEY` attributes does not have a value when the choice card is invoked, the `DEFAULT` and `IDEFAULT` attributes are used to specify the default item by name or index, respectively. For example, the last HDML deck used `the DEFAULT=zip` attribute to indicate that, by default, the second item should be chosen. This could also have been done by providing the attribute `IDEFAULT=2`.

Choices are specified using the `<CE>` tag. At least one `<CE>` tag must be included in each `<CHOICE>` tag, although normally, of course, there are at least two. The `<CE>` tag specifies how the choice will be labeled, along with its value and, optionally, a task to be performed if it is selected.

Most `<CE>` tags are simple, like the one shown in the previous city and zip code example. The `VALUE` attribute of a `<CE>` tag specifies the value assigned to the `KEY` variable when the user makes his or her choice. This value can be used elsewhere in the stack or posted to the server where the content originates to be processed.

More complex `<CE>` tags contain a `TASK` attribute. The `<CE>` tag accepts all of the same attributes used to modify task behavior that the `<ACTION>` tag does (see Table 10-4). For example, the following tag causes the browser to navigate to another deck named on the current origin server if the user selects the option labeled "football".

```
...
<CE VALUE="football" TASK=GO DEST="football.hdml">
  <IMG ICON="football"><TAB>Football News
...
```

The Entry Card

The `<ENTRY>` tag creates a card, commonly called an *entry card,* with a single entry field. (See Figure 10-6 for an example.)

Input from the user may be numeric, alphanumeric, or constrained to certain kinds of characters. Like the `<CHOICE>` tag, the `<ENTRY>` tag declares a separate card and accepts all of the same attributes as does the `<DISPLAY>` tag. This card prompts the user for a zip code, then the browser downloads a new deck created by the Common Gateway Interface (CGI) `weather` on the remote server using that zip code:

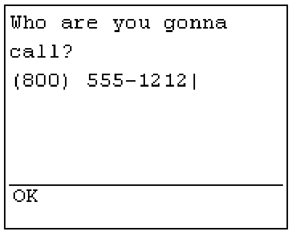

```
Who are you gonna
call?
(800) 555-1212|
OK
```

Figure 10-6. An entry card

```
<ENTRY MARKABLE=FALSE NAME="2" KEY=zip FORMAT=NNNNN>
  <ACTION TYPE=ACCEPT TASK=GO DEST="weather?zip=$zip" >
  Zip:
</ENTRY>
```

The `<ENTRY>` tag uses the `KEY` and `DEFAULT` attributes in the same manner as the `<CHOICE>` tag does. The `KEY` attribute specifies the destination variable for the input, and the `DEFAULT` attribute specifies a default value in the event that the destination variable was empty when the entry card was drawn.

By default, an entry card's action will be performed and the browser will navigate to a subsequent card only if the user provides some input. The `EMPTYOK` attribute—if set to `TRUE`—indicates that an action or that navigation may occur even if no input is made.

The `NOECHO` attribute prevents entered data from being displayed. The use of this tag is a matter of some debate, however. Many developers feel it is appropriate to disable the display for sensitive input such as passwords or access codes, while others argue that a phone display's small size makes it easy for the user to enter sensitive information discreetly, and that not echoing input detracts from a deck's user interface. For those who choose to use it, the tag is there. By default, the

NOECHO attribute is FALSE, but when it is set to TRUE, entered characters will not be displayed to the user.

The FORMAT attribute is the most important <ENTRY> attribute you'll use. It defines a *format specifier*, which is used by the HDML browser to validate input before allowing the user to continue. This specifier can include the following:

- **Integers**, which specify the number of characters that should match a specific type.
- **Asterisks**, which indicate that an arbitrary number of characters should be allowed.
- **One-letter type declarations**, which indicate the permitted type(s) of input. (Possible type declarations are listed in Table 10-6.)

CHARACTER	ALLOWED INPUT TYPE
A	Any symbolic or uppercase alphabetic character.
a	Any symbolic or lowercase alphabetic character.
M	Any symbolic, numeric, or uppercase alphabetic character (changeable to lowercase); for multiple character input, defaults to uppercase first character.
m	Any symbolic, numeric, or lowercase alphabetic character (changeable to uppercase); for multiple character input, defaults to lowercase first character.
N	Any numeric character.
X	Any symbolic, numeric, or uppercase alphabetic character (not changeable to lowercase).
x	Any symbolic, numeric, or lowercase alphabetic character (not changeable to uppercase).

Table 10-6: Character Type Declarations for HDML Input Specifiers

For example, if the required input was a United States zip code, you could use the input specifier 5N to restrict allowable input to exactly five numeric characters; for a United States two-letter state code, you might make the input specifier 2A.

Characters in the format specifier can also be used to format user input. For example, the input specifier \(NNN\)\ NNN\-NNNN accepts a phone number with area code. The display will begin with a leading parenthesis. Once three digits have been entered, a closing parenthesis will be displayed. After another three digits have been entered, a hyphen is displayed, and then the user can enter the final four digits.

The following card shows this input specifier in action, prompting the user for a phone number:

```
<HDML VERSION=3.0>
<ENTRY KEY=phone FORMAT="\(NNN\)\ NNN\-NNNN">
  <ACTION TYPE=ACCEPT TASK=CALL NUMBER=$phone>
  <ACTION TYPE=SEND TASK=CALL NUMBER=$phone>
  Who are you gonna call?
  </ENTRY>
</HDML>
```

Integrating Actions and Input

Actions are closely related to input. If HDML did not make it possible send user input to servers for processing, then the input would have little purpose. Similarly, actions are mostly useful for purposes of navigation based on user choices.

HDML variables provide the link between actions and input. Through variables, user input is collected and verified; these variables are then sent to the origin server through an action for processing. Origin servers can format their responses as simple HDML decks, or use variables again to enable dynamic content display. HDML's use of variables is, in a way, similar to that of programming languages, a fact best demonstrated by means of an example.

The weather example we've been using throughout this chapter uses variables to return the user's location to the origin server. Both the city/state and zip code inputs use the hidden result card to request the weather results. This hidden card merges the variables entered by the user and posts them to the origin server for processing:

```
<HDML VERSION=3.0>
<DISPLAY NAME="home" TITLE="Weather" MARKABLE=TRUE>
  <-- Home card display here -->
</DISPLAY>

<CHOICE MARKABLE=FALSE NAME="InputChoice"
  KEY=zip IKEY=num IDEFAULT=2>
  <ACTION TYPE=ACCEPT TASK=GO DEST="#$num">
  Enter your destination by:
  <CE VALUE=city>City & State
  <CE VALUE=zip>Zip code
</CHOICE>

<ENTRY MARKABLE=FALSE NAME="1" KEY=city FORMAT=*M>
```

```
  <ACTION TYPE=ACCEPT TASK=GO DEST=#state>
  City:
</ENTRY>

<ENTRY MARKABLE=FALSE NAME="state" KEY=state FORMAT=AA>
  <ACTION TYPE=ACCEPT TASK=GO DEST=#result>
  State:
</ENTRY>

<ENTRY MARKABLE=FALSE NAME="2" KEY=zip FORMAT=NNNNN>
  <ACTION TYPE=ACCEPT TASK=GO DEST=#result>
  Zip:
</ENTRY>

<NODISPLAY NAME="result">
 <ACTION TYPE=ACCEPT TASK=GO
  DEST="http://masamune.lothlorien.com/wx"
  METHOD=POST
  POSTDATA="choice=$numcity&city=$city&state=$state&zip=$zip">
</NODISPLAY>
</HDML>
```

The action for the hidden card result collects the user's input into a single POST request sent by the client to the server. The server-side script then uses the values provided to create and return a deck with the desired information.

In the last example, the deck's interaction with the server occupies its own card. The deck could also have been written with the server interacting over several cards, as shown here:

```
...
<ENTRY MARKABLE=FALSE NAME="state" KEY=state FORMAT=AA>
<ACTION TYPE=ACCEPT TASK=GO
  DEST="http://masamune.lothlorien.com/wx"
  METHOD=POST
  POSTDATA="choice=$numcity=$city&state=$state&zip=$zip">
  State:
</ENTRY>
<ENTRY MARKABLE=FALSE NAME="2" KEY=zip FORMAT=NNNNN>
<ACTION TYPE=ACCEPT TASK=GO
  DEST="http://masamune.lothlorien.com/wx"
  METHOD=POST
  POSTDATA="choice=$numcity&city=$city&state=$state&zip=$zip">
  Zip:
</ENTRY>
</HDML>
```

Although this version may make more sense at first and may be slightly faster to create, the approach chosen in the original answer is easier to debug and maintain. If you keep input cards separate from navigation, you'll only need to change a single card if your origin server URL changes.

HDML User Interface Design

One benefit of HDML is how relatively consistent the hardware on which it is available is. Most phones running the Phone.com browser share a large number of characteristics, which together represent the notion of an *abstract phone*, including the following:

- A fixed-width display, at least four lines deep and twelve characters across.
- Support for vertical scrolling.
- Support for the ASCII character set (upper and lower case).
- Numeric and alphanumeric character entry and editing.
- Choice selection via an arrow keypad, numeric keys, or rocker switch.
- Keys providing functionality for ACCEPT and PREV (to move forward and back between screens).
- One or two programmable soft keys with LCD labels.

At a minimum, all wireless handsets that can browse HDML share these characteristics. A few may have more features, but if you are developing content for these devices you shouldn't assume a higher level of functionality than what is outlined here.

In addition to the constraints of the handheld client, it is possible for the the UP.Link server to impose additional restrictions of its own. Most important, current versions of the UP.Link server cannot transmit HDML decks larger then 1492 bytes to browsers. Decks larger than this generate an error in the UP.Link server. Because the server's manipulation of a deck can change its size slightly, decks slightly smaller than this limit may also fail. To be safe, keep all uncompiled decks under 1200 bytes. If this challenge sounds onerous, bear in mind that 1KB of data on a screen phone is quite a bit—it can represent up to 21 screens of content!

Cards and Decks

Wireless handsets are not suitable tools for browsing long passages of text. Long runs of text are best avoided in HDML documents (as in WML ones), but if they are necessary, they should be broken up and presented on several different cards. When the phone shows multiple cards, the default key should move the user to the next card, and navigating backward should be accomplished by the handset's PREV key, not by a soft key.

As with HDML (and even wireless HTML), less is truly more. A deck should be simple, and content kept brief. Screens are small and bandwidth precious, making brevity necessary.

Tasks and Actions

The functions to be performed by soft keys can be assigned within HDML content. By convention, the first soft key should always be assigned to the default option, and the second one reserved for less common operations, such as "Menu," "Delete," or "Home." (One secondary action that doesn't belong on a soft key is a link to user help—use HDML's HELP action.) When many functions are available at one time, the best solution is to label the second soft key simply "Menu" and offer the user a list of choices.

Several hardware keys on a Phone.com phone can be overridden. When reassigning built-in keys, think carefully about what the user will expect when using your deck. Assign keys in such a way as to extend and integrate the phone's functionality with your deck, rather than to try to tailor its behavior to meet your expectations. Rewiring the phone's SEND button, which ordinarily is used to make a call, to display a help message will not only be difficult to use but outright annoying to most users. A better use of the SEND key, for example, is to map it to dial the number currently displayed on a card.

Data Input

HDML content must enable data entry, of course, but keep in mind that data entry on wireless phone handsets is always time-consuming and frustrating. Limit users' need to enter data by providing lists of choices wherever possible. Even these lists should be brief, preferably including less than seven items each.

Where text entry is unavoidable, use format specifiers to restrict input to the kinds of characters required. For example, since it makes no sense for a user to enter alphabetic characters in a United States zip code, it's best to use format specifiers to restrict entry to digits to prevent users from entering a letter in error.

Displaying Images

As not all HDML devices can display images, images should be used sparingly if at all in HDML-based content. Include an image in a deck only if it carries a meaning that text cannot. Every image should be accompanied (using the `ALT` attribute) by a text legend detailing its meaning. These guidelines are identical to the guidelines for using images with WML and HTML content.

Bookmarks

Bookmarks provide a quick way for users to return to a commonly viewed card—just as desktop-browser bookmarks enable easy access to commonly viewed pages.

Dynamic content, however, should not be bookmarked. If a page was generated using input from a form, for example, you would want to make only the beginning of the form interface markable, not the end results—this way, you can ensure users don't make the mistake of bookmarking the wrong page.

Summary

HDML provides a rich alternative to HTML for content intended for screen phones and similar devices. While HDML's organization and syntax predate WML, the two languages share many concepts, making it easy for developers to create both HDML and WML content for the same service.

Like WML, HDML presents content in decks of cards. Cards may contain text, choice lists, or user input. Navigation between cards and other activities take the form of tasks, which can be bound to interface controls such as the phone's ACCEPT key or a soft key. Using tasks and variables that contain strings, decks can collect and validate input from a user, perform simple operations, and display the results.

While HDML is a flexible language capable of some complexity, it is best to keep decks simple. Users usually interact with wireless devices, with one hand, in a variety of settings that divide their attention among several tasks. With this in mind, you should organize decks according to the activities they help a user to perform, and provide only the information that is necessary during a particular transaction.

Don't use images unless you're sure they're worth the size they occupy. When choosing an image, be sure to see if the phone provides one you can use through the `<IMG> LOCAL` attribute. And, of course, avoid user input unless it's absolutely necessary.