There are two ways to obtain ROM files for PalmOS devices:

- Download a ROM image from the Palm Resource Pavillion
- Upload a ROM image from a PalmOS device to our computer

Palm provides ROM images for free, but they do require us to sign up as a Palm Developer. They provide ROMs for many different types of PalmOS devices; even if we have an older, monochrome model we can still see what our HAWHAW pages will look like on a color device.

For more information about obtaining PalmOS ROMs or uploading them from our PDA, visit the Palm developers' web site at http://www.palm.com/developers/.

Refer Appendix H for detailed instructions to set up Palm Desktop, POSE, and AvantGo on a Windows desktop. UNIX users will have to go through the AvantGo sign-up and installation process to get the necessary `.PRC` files for the POSE.

Delivering Custom Content with PHP

Now that we're familiar with the platforms we'll be developing for, we need to think about how we can best use PHP to create an application that will work with both WAP WML, and PDA-optimized HTML. We already know that we can use GD to resize images and save them in both WBMP and JPEG format (we need to do it for our coffee shop photos). The tricky part now is finding a way to tailor our scripts' output to two different platforms.

Determining the User Agent

The first thing we need to know is the browser that is requesting our script; this will tell us which type of markup – WML or HTML – our scripts need to return.

This is easy enough to determine by accessing PHP's built-in `$_SERVER` array. Specifically, we need to examine the contents of `$_SERVER["HTTP_USER_AGENT"]`. We'll also need a list of user agents and the `HTTP_USER_AGENT` header they send, so we know which user agents expect which kind of content.

WML/HTML on the Fly

The next step will be to figure out a way to take information from the database, and generate WML or HTML pages on the fly. We could come up with an elaborate system of headers and footers, or separate sets of templates for WML and HTML pages, but then we'll be locked into a fairly static format and we'll end up updating two different files any time we want to adjust the page layout.

On top of that, we've only just scratched the surface of WML; mixing PHP code and WML templates is a less than ideal situation when we're not very familiar with WML alone.

Does this sound like a headache yet? Fortunately there's another option that will make our lives a lot easier.

HAWHAW

HTML and WML Hybrid Adapted Webserver (HAWHAW) is a toolkit that makes developing cross-platform mobile applications a lot easier than developing a proprietary template system. Even better, it creates applications that are compatible with platforms other than WAP and PDA-friendly HTML. In addition to WAP and HTML, HAWHAW supports **HDML** (a WML predecessor), **i-Mode** (i-Mode is the packet-based service for mobile phones offered by NTT DoCoMo, Japan), and **MML** (Multimedia Markup Language).

i-Mode, since it was developed by a Japanese wireless communications company, is mostly used in Japan. MML is another wireless markup language widely used in Japan, and was developed by J-Phone Communications Co Ltd.

The HAWHAW toolkit comes with several components:

- **HAWHAW XML**
  HAWHAW XML is the markup language used by the HAWHAW toolkit. It is similar in appearance to WML but is somewhat simplified. We can think of HAWHAW XML as an abstraction layer between our content and the different platforms that HAWHAW supports.
- **HAWXY**
  HAWXY is a proxy server in the form of a PHP script that translates HAWHAW XML into the appropriate markup for any user agent requesting that document, based on the `$_SERVER["HTTP_USER_AGENT"]` variable.
- **The HAWHAW PHP class library**
  The HAWHAW PHP class provides a set of functions for building standalone mobile applications that don't need to use HAWHAW XML or HAWXY. It generates the appropriate markup on its own. We'll be using the HAWHAW PHP class extensively in our Coffee Shop Finder mobile application.

Object-Oriented Web Pages

The HAWHAW PHP class uses an object-oriented (OO) approach for building the pages of our application. At the beginning of each script, we create a `HAW_deck` object, which has several properties and methods associated with it. Page elements contained within the `HAW_deck` are also represented by objects, which in turn have their own properties and methods for setting attributes, and adding child objects.

Here's what a simple script using the HAWHAW class looks like:

```
<?php

include_once("hawhaw.inc");

$haw = new HAW_deck("My HAWHAW Deck");

$hello = new HAW_text("Hello, World!");

$haw->add_text($hello);

$haw->create_page();
?>
```

That's all there is to it. The HAWHAW class takes care of figuring out what type of user agent is requesting the page, and transforms the HAW_deck object into the correct markup. Here's what our 'Hello, World!' script returns when called from a WAP device:

```
<?xml version="1.0"?>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

<!-- Generated by HAWHAW V4.06 (C) Norbert Huffschmid -->

<wml>
  <card title="My HAWHAW Deck">
    <do type="prev" label="Back">
     <prev/>
    </do>
    <p>
    Hello, World!
    <br/>
    </p>
  </card>
</wml>
```

And here's what the same script returns when called by AvantGo:

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">

<html>

  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <meta name="GENERATOR" content="HAWHAW V4.06 (PHP) (C) Norbert
        Huffschmid">
    <meta name="HandheldFriendly" content="True">
    <title>My HAWHAW Deck</title>
  </head>

  <body>
    <div align="left">
     Hello, World!
     <br>
    </div>
  </body>

</html>
```

The HAWHAW class methods and properties are intuitive and consistent, as we'll see when we start building our application.