Published on **Wireless DevCenter** (http://www.oreillynet.com/wireless/)
http://www.oreillynet.com/pub/a/wireless/2004/02/20/mobile_browsing.html
See this if you're having trouble printing code examples

Creating Web Content for Mobile Phone Browsers, Part 2

by Robert Jones
02/20/2004

In part one of this series, I took you inside the WML and XHTML MP markup languages, highlighted the differences between them and showed how to serve the pages from an Apache web server. In part two, I want to show you how we deal with the diversity of phones out there, and how we can tailor our content to communicate effectively with those phones.

This diversity is the curse that afflicts most of the WAP sites in use today. Aiming for as large an audience as possible, sites are forced to use only the most basic features of WML, which is already one of the more basic markup languages. The resulting web pages are dull and completely ignore the capabilities of newer phones. But those new features, in particular the larger color screens, are exactly the reason people are buying new phones!

How do we resolve this? Even if we give up on older phones and just deliver XHTML MP, we still have to deal with a range of screen sizes and browser features. It's a mess. But with a little bit of extra work, there are several approaches that can move us forward.

In any request for a page from a web server there are two pieces of information we can use. The `HTTP_ACCEPT` header contains the MIME types that a browser can accept. This tells us the TYPE of information that we can return. The `HTTP_USER_AGENT` header tells us the identity of the browser and, typically, the phone. That alone does not do us a lot of good, but with a database that maps identity to things like screen size then we can infer how to best display our content.

Server Redirection

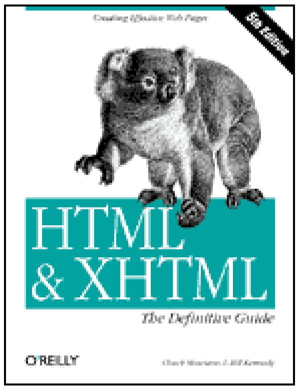
Our first approach will use the `HTTP_ACCEPT` header of a request to figure out if a browser is a mobile phone and whether it can handle XHTML MP or just WML. All the action happens in the web server itself, so this is a great solution for static web pages.

It also addresses another problem, in that we want to avoid entering long URLs into a mobile browser at all costs. For most sites the best solution would be for a user, mobile or not, to simply go to the home page. With server redirection we can detect the type of browser and return the regular home page to desktop browsers but redirect mobile browsers to a simpler page of contact information. With the Apache web server, we accomplish this using the URL rewriting capabilities of the mod_rewrite module. This is a superb toolkit for manipulating URLs but it is complex, and so we'll only present the minimum information needed for our present purpose.

Take a look at your Apache configuration file (`/etc/httpd/conf/httpd.conf`) and look for these two lines. They should both be there but you might need to un-comment them. These load the rewrite functions into the Apache.

```
LoadModule rewrite_module  modules/mod_rewrite.so
AddModule mod_rewrite.c
```

Related Reading



HTML & XHTML: The Definitive Guide, 5th Edition
By Chuck Musciano, Bill Kennedy

Table of Contents
Index
Sample Chapter

Read Online--Safari
Search this book on Safari:

Next we need a Directory block that defines what happens when a specific page is requested. Set the directory path and filenames to those appropriate to your site and enter all the statements. Rewrite statements work by defining a set of conditions and a rule that is applied to files that match the conditions. The first 'block' of conditions and rules that are matched to a browser request are used and subsequent blocks are skipped.

Go
Only This Book
☐ Code Fragments only

```
<Directory "/var/www/craic_external/html/mobile/ora">

  # This enables rewriting in this directory

  RewriteEngine On

  # Catch XHTML MP browsers first and return XHTML MP content to them

  # Our conditions are that the browser must accept XHTML MP and WML
  # content. If we just use the xhtml+xml rule then we catch Mozilla
  # browsers by mistake.
  # There is an implicit AND between the two conditions.

  RewriteCond %{HTTP_ACCEPT} application/xhtml\+xml
  RewriteCond %{HTTP_ACCEPT} text/vnd\.wap\.wml

  RewriteRule index.html$ index.xhtml [L]

  # This next block will catch browsers that only understand WML

  RewriteCond %{HTTP_ACCEPT} text/vnd\.wap\.wml

  RewriteRule index.html$ index.wml [L]

  # Browsers that match neither block, such as regular screen
  # browsers, could be caught by a final rewrite rule placed here,
  # or we could leave it out and have nothing happen to the
  # requested URL. This is the default.

</Directory>
```

If a browser passes the URL http://www.craic.com/mobile/ora Apache will add 'index.html' to it by default. If that request comes from a XHTML MP browser then index.html will be rewritten to index.xhtml and that file will be returned. If the browser can only handle WML the request is rewritten to index.wml and if the browser is 'conventional' then nothing happens and index.html is returned. This is the block that controls the directory on my web site that contains all these examples. Access it from a mobile browser and you get the WML or XHTML MP pages displayed. Use a desktop browser and you will get links to all the source code.

CGI Scripts

Server redirection is easy to set up and great for handling static web pages. But when you have dynamic content, from Perl CGI script for example, it makes more sense to have the script figure out what the browser can accept and switch accordingly. Here is a simple CGI script that echoes the variables passed in a request back to the browser in the form of a HTML, WML or XHTML MP page as appropriate.

```
#!/usr/bin/perl -w

# mime_types.cgi

# Return the list of MIME types that the browser can handle
# in the appropriate markup language - HTML, XHTML MP or WML

use CGI;
my $cgi = new CGI;

my %accept = ();
```

```
foreach my $type (split /\s*\,\s*/, lc $cgi->http('HTTP_ACCEPT')) {
    $type =~ s/\s+//g;
    $type =~ s/\;.*$//;
    $accept{$type} = 1;
}

my $language = 'html';
if(exists $accept{'text/vnd.wap.wml'}) {
    $language = 'wml';
    if(exists $accept{'application/xhtml+xml'}) {
         $language = 'xhtml_mp';
    }
}

if($language eq 'wml') {
    respond_wml();
} elsif($language eq 'xhtml_mp') {
    respond_xhtml_mp();
} else {
    respond_html();
}

exit;

#---------------------------------------------------------------

sub respond_html {

   print qq[Content-type: text/html\n\n];

   print qq[<html><head><title>Browser Info - HTML</title></head>\n];
   print qq[<body bgcolor="white">\n];
   print qq[Browser Information - HTML<br>\n];

   foreach my $type (sort keys %accept) {
      print qq[$type <br>\n];
   }

   print qq[</body></html>\n];
}

#---------------------------------------------------------------

sub respond_xhtml_mp {

   print qq[Content-type: application/xhtml+xml\n\n];

   print qq[<?xml version="1.0"?>\n];
   print qq[<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.0//EN"\n];
   print qq["  http://www.wapforum.org/DTD/xhtml-mobile10.dtd">\n];

   print qq[<html xmlns="http://www.w3.org/1999/xhtml">\n];
   print qq[<head><title>XHTML MP</title></head>\n];
   print qq[<body bgcolor="white">\n];
   print qq[Output as XHTML MP<br/>\n];

   foreach my $type (sort keys %accept) {
      print qq[$type <br/>\n];
   }

   print qq[</body></html>\n];
}

#---------------------------------------------------------------

sub respond_wml {
   print qq[Content-type: text/vnd.wap.wml\n\n];

   print qq[<?xml version="1.0"?>\n];
   print qq[<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" ];
   print qq["http://www.wapforum.org/DTD/wml_1.1.xml">\n];

   print qq[<wml>\n];
   print qq[<card id="main" title="WML">\n];
   print qq[<p>Output as WML<br/>\n];

   foreach my $type (sort keys %accept) {
      print qq[$type <br/>\n];
   }

   print qq[</p></card></wml>\n];
}
```

You can access this example at: http://www.craic.com/cgi-bin/mobile/ora/mime_types.cgi

Choosing between WML and XHTML MP for the markup and then returning a page that makes modest use of the features of each language seems like it could be the right level of effort for most WAP developers. It allows any WAP user to access our content but lets us apply some of the richer design features in XHTML MP where appropriate.

But the approach still does not tell us anything about the device that is making the request. We can get the identity of that from the `HTTP_USER_AGENT` header, but we need to know what that phone/browser is capable of displaying before we can really tailor our content. We want information like the screen size and depth and the navigation keys that are available to us. What we need is a database of information about every phone on the market that we can interrogate and use to tailor our response to a specific device.

The WURFL

The WURFL project is a brave attempt to do exactly this, with the database taking the form of a XML file. The WURFL is an open source effort, guided primarily by Laszlo Nadai, Luca Passani and Andrea Trasatti but relying on the community at large to ensure it is complete and accurate. Currently it tracks more than 100 features across more than 400 devices. That is a lot of information to maintain or to search through. The creators of the WURFL were smart and realized two important things. First is that contemporary phones from a given manufacturer tend to be very similar to each other, so that for many purposes all you need is a single profile for that family of devices. Second is that most developers will not tailor their response down to the fine detail of each phone. Just knowing the main feature set and the screen size is sufficient for most of us.

Accordingly the WURFL structures its data into a hierarchy that has a generic profile at its root. This allows developers some flexibility to choose what level of detail they want to deal with as well as the possibility to infer the features of a phone that is not in the WURFL, based on part of the text in the User Agent header.

The innards of the WURFL itself are not for the faint hearted. Fortunately there are access libraries for Perl, PHP, .Net and Java. The following code illustrates its use in Perl using the WURFLLite.pm module from Rainier Hillebrand, available at http://webcab.de/WURFL.

```
#!/usr/bin/perl -w -I.

# wurfl.cgi

# Based on the programming examples for WUFLLite.pm written
# by Rainer Hillebrand, rainer.hillebrand@webcab.de.
# The latest version can be found at:  http://webcab.de/WURFL/

use CGI;
use WURFLLite;

my $cgi = new CGI;
```

```perl
my $ua = $cgi->http('HTTP_USER_AGENT');

my %wurfl;
my $content = '';

open INPUT, "< ./wurfl.xml";
while (<INPUT>) { $content .= $_;}
close INPUT;

parseXML(\%wurfl, \$content);

# Search for the User Agent and return the device ID

my $best_device_id = '';
my $best_user_agent = '';
my $best_ua_length = 0;

my $device_id = '';

foreach my $device_id (keys %{$wurfl{'devices'}}) {

   my $dev_user_agent =
     $wurfl{'devices'}->{$device_id}->{'attributes'}->{'user_agent'};

   if(not defined $dev_user_agent or $dev_user_agent eq '') {
         next;
   }

   if( $ua =~ m|^\Q${dev_user_agent}\E| and
       length($dev_user_agent) > $best_ua_length) {
       $best_device_id  = $device_id;
       $best_user_agent = $dev_user_agent;
       $best_ua_length  = length($dev_user_agent);
   }
}

$device_id = $best_device_id;

if (not defined %{$wurfl{'devices'}->{$device_id}}) {
    # Device not found
    exit;
}


# Build the list of fallback device ids

my @device_ids = ($device_id);
my $dev = $device_id;

while ($dev = $wurfl{'devices'}->{$dev}->{'attributes'}->{'fall_back'}
       and $dev ne 'root') {
   push @device_ids, $dev;
}


# Build a hash of the capabilities of the device using
# the fallback list, starting with the most generic

my %caps = ();

while ($dev = pop(@device_ids)) {

    my $dev_caps = $wurfl{'devices'}->{$dev}->{'capabilities'};

    foreach my $group ( keys (%{$dev_caps})) {
        foreach my $cap ( keys (%{${dev_caps}->{$group}})) {

           $caps{$group}->{$cap} = ${dev_caps}->{$group}->{$cap};
        }
    }
}


# Return the information as WML

print qq[Content-type: text/vnd.wap.wml\n\n];

print qq[<?xml version="1.0"?>\n];
print qq[<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
         "http://www.wapforum.org/DTD/wml_1.1.xml">\n];

print qq[<wml>\n];
print qq[<card id="main" title="Wurfl test">\n];
print qq[<p>WURFL Test - WML<br/>\n];

print qq[<b>Device ID</b> $device_id <br/>];
print qq[<b>User Agent</b> $ua <br/>];

print qq[<b>Rows</b>] . $caps{'display'}->{'rows'} . qq[<br/>];
print qq[<b>Cols</b>] . $caps{'display'}->{'columns'} . qq[<br/>];

my $height = $caps{'display'}->{'resolution_height'};
my $width  = $caps{'display'}->{'resolution_width'};

print qq[<b>Display Height</b>] . $height . qq[<br/>];
print qq[<b>Display Width</b>]  . $width . qq[<br/>];

print qq[<br/>\n];
print qq[<img src="/mobile/ora/line.wbmp" alt="line"
         width="$width" height="10"/>];

print qq[</p></card></wml>\n];
exit;
```

This script finds the best match in the WURFL to your User Agent and then constructs the 'fallback' list of more generic feature profiles. It then builds a hash of the specific capabilities of your phone using those profiles. To keep the example short I only return the information in WML and limit the data to the screen size in pixels and characters. It uses the screen width to scale an image of a line so as to span the entire screen of your phone.

In principle, the WURFL is a great idea but the reality leaves something to be desired. For example, my Nokia 3650 has a physical screen width of 176 pixels and the WURFL correctly tells me this, but in reality the browser can only access the center 171 pixels. It also tells me, incorrectly, that the accesskey functions don't work. It represents a step forward, just don't rely too heavily on the more detailed information it returns.

If you only have access to a single phone but want to experiment with the WURFL, then you can use the simulators. Both the applications let you specify the user agent they pretend to be. They don't take on the true feature set of that agent, in terms of screen size, etc., but they can be invaluable for testing your server side applications.

You might think that the wireless industry would have set up something like this themselves and indeed they have, in the form of the User Agent Profile (UAProf). This is one of the detailed and, in my personal opinion, horribly convoluted and acronym-ridden specifications that are managed by the Open Mobile Alliance (OMA). The WURFL folks include a link to an OMA white paper on the UAProf but, unfortunately, "Open" is a relative term and this document is only accessible to OMA members. You can see what the UAProf file for the Openwave simulator looks like at this URL: http://devgate2.phone.com/uaprof/OPWVSDK62.xml

Final Thoughts

The audience for the Mobile Internet is growing rapidly. New phones have the capability to display rich, interesting and attractive content. But the majority of the content that's available today ignores those features completely.

The diversity and rapid evolution of phone hardware is the source of the problem. In order to reach the widest audience, most sites have opted to support the lowest common denominator with web pages that use only basic text. Unless we overcome this limitation and deliver richer content, then the mobile Internet will not live up to its potential. Content that is tailored to different classes of browser is one way to move the field forward. Hopefully the examples in this article will spark your interest in this area and help you get started developing your own pages.

Links

- WML Language Reference
- XHTML MP Language Reference
- OpenWave browser simulator and other resources
- Nokia browser simulator, editor and other resources
- Apache Server Redirection
- WURFL
- The examples used in this article are available here. If you access it with a desktop browser you can retrieve the source to all the examples in this article. If you use a mobile browser you will be redirected to the appropriate WML or XHTML MP pages.

Robert Jones runs Craic Computing, a small bioinformatics company in Seattle, which provides advanced software and data analysis services to the biotechnology industry. He was a bench molecular biologist for many years before programming got the better of him.

Return to the Wireless DevCenter

Copyright © 2004 O'Reilly Media, Inc.