Horde Configuration

OSCON

July 25, 2002. San Diego

Chuck Hagenbuch <chuck@horde.org>

Introduction

The Horde Application Framework and its applications have a large number of configuration files. Users frequently have questions about which they need to edit, what configuration options, do, etc.

We'll try and hit the most interesting and useful bits.

Config Files

Every configuration file, in Horde and its applications, is distributed as filename.php.dist.

We've got a lot of config files, just for Horde:

o horde.php

o html.php

o mime_drivers.php

o mime_mappings.php

o motd.php

o nls.php

o prefs.php

o registry.php

Many of these are the same for every app once we've explained them for Horde.

Preferences

horde.php is the main Horde configuration file. This is where you configure global options, like the debugging level, and all of the global Horde backends, such as preferences, logging, a mailer, and optional things like the VFS and a Category backend.

You can store preferences in either a SQL database, an LDAP server, or only keep them for the length of a user's session. Here's a SQL configuration:

```
<?php // A SQL example
$conf['prefs']['driver'] = 'sql';
$conf['prefs']['blurbms'] = array('phptype' => 'odbc',
                                  'hostspec' => 'LocalServer',
                                  'username' => 'horde',
                                  'password' => 'password',
                                  'database' => 'horde',
                                  'table' => 'horde_prefs');
?>
```

A basic LDAP configuration might look like this. However, you are likely going to need to consult additional documentation to use LDAP, since setting up the necessary schema, for example, is non-trivial.

```
<?php // An LDAP example
$conf['prefs']['driver'] = 'ldap';
$conf['prefs']['blurbms'] = array('hostspec' => 'ldap.example.com',
                                  'port' => 389,
                                  'basedn' => 'dc=example, dc=com',
                                  'uid' => 'account_id');
?>
```

Session-based preferences are simple and useful for demo sites or pure-kiosk sites.

Configuring Any Horde SQL backend:

```
<?php // A sessions example
$conf['prefs']['driver'] = 'session';
$conf['prefs']['blurbms'] = array(); // There are no required parameters.
?>
```

SQL configuration - for Turba addressbookes, categories, the VFS, Whups, anything - all follows the same pattern as the prefs example above. Let's look at it again:

```
<?php // A SQL example
$conf['auth']['driver'] = 'sql';
$conf['auth']['blurbms'] = array('phptype' => 'odbc',
                                 'hostspec' => 'LocalServer',
                                 'username' => 'horde',
                                 'password' => 'password',
                                 'database' => 'horde',
                                 'table' => 'horde_users');
?>
```

Note that we've changed two things: it's $conf['auth'], not $conf['prefs'], and we've changed the table name. This is what you'll need to do for any Horde SQL backend - and ALWAYS double check those two changes.

html.php

html.php is where most of the look and feel of Horde - fonts, colors, sizes, etc. - is defined. Horde uses all CSS for defining layout, so in this file, you can control the look of pretty much anything. It is simply an associative array, which gets parsed by a script into CSS which is then fed to the browser. An example line looks like this:

```
<?php
$css['body']['font-family'] = 'Geneva,Arial,Helvetica,sans-serif';
```

This would get turned into the following CSS code:

```
body { font-family: Geneva,Arial,Helvetica,sans-serif; }
```

Individual applications have their own html.php files, and they override the general Horde one. This means that each application can add or change styles as necessary, but otherwise, Horde's html.php controls the look of everything.

Finally, note that the CSS generation script, by default, asks for the CSS to be cached by the browser, so it's not requested on every page load. You can turn this off with a config switch in horde.php, but it's just something to be aware of when testing changes - you might have to clear your browser's cache.

mime_drivers.php

Horde's MIME_Viewer classes provide nice viewing of a lot of different MIME types, and are used all over Horde.

The $mime_drivers_map determines which viewers are enabled Taking a viewer out of this array disables it; that type will then be handled by a more generic driver.

```
<?php
$mime_drivers_map['horde']['registered'] = array(
    'php', 'enscript', 'tgz', 'rar', 'msword', 'msexcel', 'mspowerpoint',
    'vcard', 'zip', 'rpm', 'deb', 'enriched', 'smime', 'images', 'tnef');
```

Icons for any MIME types that are handled by the default viewer (which passes data through unchanged) can be changed or added to in the Horde default viewer configuration:

```
<?php
$mime_drivers['horde']['default']['icons'] = array(
        'default'                       => 'text.gif',
        'application/x-gzip'            => 'compressed.gif',
        'application/pdf'               => 'pdf.gif',
// ... etc.
```

The MSWord document viewer is an example of one which requires an external application (in this case, wvHtml)

```
<?php
/**
 * MS Word driver settings
 * This driver requires wvWare (www.wvware.com) to be installed.
 */

/* Location of the wvHtml binary. */
$mime_drivers['horde']['msword']['location'] = '/usr/bin/wvHtml';
$mime_drivers['horde']['msword']['inline'] = true;
$mime_drivers['horde']['msword']['handles'] = array(
    'application/msword',
    'text/rtf',
    'x-extension/doc',
    'x-extension/rtf');
$mime_drivers['horde']['msword']['icons'] = array(
    'default' => 'msword.gif');
```

mime_mapping.php

You should never have to edit this file, but it's good to know it's there and what it's used for. It provides a mapping from file extensions to MIME types, such that if we get a file from somewhere - like an FTP server in Gollem - with no MIME type, we can guess the correct one. The array looks like this:

```
<?php
$mime_extension_map = array(

'Z'  => 'application/x-compress',
'ai'  => 'application/postscript',
// ...
```

Any file extension that we don't find a match for here will be mapped to the special MIME type x-extension/ext, so that Horde can handle such files gracefully internally.

motd.php

MOTD stands for Message Of The Day, and this file is used similarly to how /etc/motd is used on UNIX systems. There is one for Horde, IMP, and Gollem.. It is shown on the Horde Summary page, below the "Welcome" text and above any application summaries, and on the login pages for IMP or Gollem.

The contents are not processed in any special way; they can be HTML, PHP code, or whatever you like. You'll at least need to add a bit of HTML markup to give the text some sensible formatting - for instance, class="light".

If config/motd.php isn't there or readable, it'll be ignored. Here's an example of an inclusion that gives users a switch between HTTP and HTTPS:

```
<?php
$SERVER_SSL_PORT = 443;
$SERVER_HTTP_PORT = 80;
$SERVER_SSL_URL = 'https://www.example.com';
$SERVER_HTTP_URL = 'http://www.example.com';

$port = $_SERVER['SERVER_PORT'];

echo '<br/ ><div align="center" class="light">';

switch ($port) {
    case $SERVER_SSL_PORT:
        echo 'You are currently using Secure HTTPS<br/>';
        break;

    case $SERVER_HTTP_PORT:
        echo 'You are currently using Standard HTTP<br/>';
        break;
}

echo '<a class="small" href="' . $SERVER_HTTP_URL . '" target="_parent">' .
_("Click here for Standard HTTP") . '</a> - <a class="small" href="' .
$SERVER_SSL_URL . '" target="_parent">' . _("Click here for Secure HTTPS") .
'</a></div>';
```

nls.php

nls.php is the core configuration file for nationalization and localization for Horde, and thus affects what languages are available, what character set those languages are displayed in, multi-lingual spell checking, and timezones.

The whole file defines two arrays, $nls and $tz, which are used throughout the framework. The language list lives in $nls['languages']; an entry looks like this:

```
<?php
$nls['languages']['ja_JP'] = 'Japanese';
```

To disable a language from being used, just remove it from this array. The $nls['aliases'] array defines what language a more general language code defaults to; for instance, $nls['aliases']['en'] = 'en_US' means that a browser requesting the 'en' language will get U.S. English.

You should not need to edit the character set list or the timezone list unless you add a translation to your installation, or find a timezone missing. The character set definition is just the character set that the translation files for that application use; for example, the Greek translation is in ISO-8859-7:

```
<?php
$nls['charsets']['el_GR'] = 'ISO-8859-7';
```

The spell checking section defines what flags we pass to ispell/aspell to request the appropriate language:

```
<?php
$nls['spelling']['pt_PT'] = '-T latin1 -d portuguese';
```

prefs.php

The Horde preferences system is extremely flexible, and once you get the hang of the prefs.php configuration format you'll find yourself dreaming up all kinds of cool things to do with it. This file has two purposes:

The format for an individual preference is pretty simple. Here is an example for the Horde 'theme' preference:

```
<?php
// color theme
$_prefs['theme'] = array(
    'value' => '',
    'locked' => false,
    'shared' => true,
    'type' => 'enum',
    'enum' => array(''           => _("Horde Standard"),
                    'brown'      => _("Brown"),
                    'green'      => _("Green"),
                    'sun'        => _("Sun")),
    'desc' => _("Select your color scheme.")
);
```

This is an 'enum' preference, which means that users are choosing from a pre-defined set of options. Other preference types include 'number' (any number), 'checkbox' (on or off, 1 or 0), 'text', 'textarea' (short and long bits of freeform text), 'link', 'special', and 'implicit'. Most of these are self-explanatory, and are checked by the automatically generated UI to make sure they're valid - a number preference will be rejected if the user types in 'A'.

'special' and 'implicit' preferences get extra explanation, and they usually go together. An implicit preference is simply one which is set elsewhere - it might be toggled by the application's normal UI, like a sorting direction, or it might be handled by a special preference. special preferences are simply placeholders for things which aren't really preferences, but which need to appear in the UI to handle setting of implicit prefs - things like a folder selection preference, where we want to allow creating a new folder on the fly, that don't fall into generic types. 'link' preferences just insert a link to another page into the preferences UI, and like special prefs, don't actually have a value.

The format for preference groups, which is what the UI is built of, is also pretty simple:

```
<?php
$prefGroups['display'] = array(
    'column' => _("Other Information"),
    'label' => _("Display Options"),
    'desc' => _("Set your color scheme, page refreshing, and other display
options."),
    'members' => array('theme', 'summary_refresh_time'));
```

This defines a preference group, or a sub-page, with two preferences on it: the theme preference and the time to refresh the summary screen. It will appear in the column labeled "Other Information"; so will any other preference groups with that same column name. The description on the main UI page is defined, as well as that specific section's label - "Display Options". The UI code automatically figures out how many columns are present and lays them out.

registry.php

The last configuration file we'll cover is registry.php, and it's also the most complicated that Horde has. Fortunately, you shouldn't need to change a single thing in it for a standard install. But, for non-standard installs, there's a lot of useful stuff to tweak, and for developers, this is a gold mine.

There are three major things defined in registry.php:

o Core application configuration stanzas

o Application services (methods

o For services with multiple implementations, which provider should be used

An application's configuration stanza defines that application's name, its icon, where its files live, whether it should show up in the Horde menu, and whether it requires authentication or is open to guests. Here's one for Jonah:

```
<?php
$this->applications['jonah'] = array(
    'fileroot' => dirname(__FILE__) . '/../jonah',
    'webroot' => $this->applications['horde']['webroot'] . '/jonah',
    'icon' => $this->applications['horde']['webroot'] .
'/jonah/graphics/jonah.gif',
    'name' => _("Headlines"),
    'allow_guests' => false,
    'show' => true
);
```

In a default installation, with Jonah installed under horde/ as horde/jonah/, the only settings you might want to tweak are 'allow_guests' and 'show'. If you set any application's show blurbmeter to false, then it won't show up in the Horde menu across the bottom of the screen. If you were to set Jonah's allow_guests setting to true, then any user would be able to view headlines without logging in. Obviously, we default to requiring authentication for every Horde app.

Horde has several special settings that you need to be aware of:

```
<?php
    'cookie_domain' => $_SERVER['SERVER_NAME'],
    'cookie_path' => '/horde',
    'server_name' => $_SERVER['SERVER_NAME'],
    'server_port' => $_SERVER['SERVER_PORT']
```

cookie_domain and cookie_path control the settings for every cookie set by Horde, including the session cookie. If you are using Horde on a cluster, then you may need to modify cookie_domain to be .domain.com, so that all servers in your cluster receive all cookies. Similarly, if you have horde installed somewhere other than /horde - for instance, as the root of your webserver - you'll need to change cookie_path; in this case, to ". server_name and server_port should only be modified if your webserver, for some reason, reports incorrect values to PHP.

Horde services are mostly beyond the scope of this talk, but here's what you're looking at:

```
<?php
$this->services['mnemo']['memos']['list'] = array(
    'file' => '%application%/lib/api.php',
    'function' => 'mnemoListMemos',
    'args' => array('sortby', 'sortdir'),
    'type' => 'array'
);
```

'file' defines what file the function defined by 'function' - i.e., mnemoListMemos() - is defined in. When this service is invoked, that file is automagically included and that function called. 'args' is how many arguments the function expects to recieve, and 'type' is the return type of the method, although that return type is currently not enforced.

Finally, the handler definitions look like this:

```
<?php
$this->registry['publickey']['add'] = 'turba';
```

All this means is that the publickey/add method will be handled by the application called 'turba' when it is called.

Wrap up

We've touched on a lot very quickly, but hopefully you now have a better sense of your way around the Horde config/ directories. Here are a few resources to keep in mind when exploring Horde:

Horde website: http://www.horde.org/

Developer Resources: http://dev.horde.org/

Presentations: http://www.horde.org/papers/

Index