



cPAddon Scripts:

Creation, Installation, and Distribution

Table of Contents

cPAddon Scripts Overview	3
Creating a New cPAddon Script	4
Basic Sample Module	5
Creating a cPAddon Upgrade	6
\$meta_info Hash Reference Table	7-9
Custom Install Fields	10
Sample Custom Install Hasref	11
Install_fields_hook	12
Config File Variables	13
Variable List	14
Invisible cPAddon Scripts	15
Example Script	16
Addon Aliases	17
Addons that Require Licenses	18
Distributing Your Addons	19-20
Installing Addons Manually	21
Licensing Your Addons	22
Creating Your cPanel Sync Server	23
cPanel Sync Module	A

cPAddon Scripts

The cPAddons system allows a script to be packaged up so that it can be installed on a user's website, upgraded, uninstalled and otherwise managed from cPanel and WebHost Manager. This manual describes technically how to package a script as a cPAddon and how to maintain that script.

Any script that you wish to install on multiple websites could potentially be a cPAddons script. A guestbook, content management system, blog engine, etc could all be cPAddons scripts. The benefit to having cPAddons scripts is that these scripts can be installed by cPanel users from their cPanel interface after the cPAddon script is installed on the server.

Creating a New cPAddon Script

You will first need to create a script that is compatible with the cPAddons system if you wish for the script to be accessible from the cPanel interface. To do so, follow the steps below:

Assuming we are making a cPAddon for a CMS called “Foo” version 1.0:

1. Create the working repository/directory
2. Put Foo_1.0.tar.gz into the working directory
3. Untar Foo_1.0.tar.gz into Foo_1.0.orig/ and Foo_1.0.work/
4. Make a directory Foo/
5. Make Foo.pm

See Basic Sample Module to see what should be in this file.

6. Modify Foo_1.0.work/ here as needed

Basically your goal here is to get all of the files necessary to the install all prepped and together.

- * Remove any install scripts and upgrade scripts since this is all handled via the cPAddons interface.
- * Prepare any config files, this entails:
 - o Adding the “Config file Variables”
 - o Adding the filename to the “config” key of the module’s \$meta_info hashref
- * Set appropriate permissions on all the files (world readable config files and world writable directories will result in a much lower security rating)

7. Create Foo/1.0.mysql if needed

The mysql file uses the “Config file Variables”

You should also include a use statement at the top. Assuming the ‘mysql’ key in the module’s \$meta_info is:

```
‘mysql’ => [‘foo’],
```

your mysql file would start like this:

```
USE [% foo %];
```

```
CREATE TABLE [% table_prefix %]_barbaz (
```

8. Create Foo/1.0.tar.gz from Foo_1.0.work/

Note: The files in that tar ball get directly untarred into the installation directory!! Keep that in mind when creating it.

9. Create a file named Foo/lisc (contains you license verbiage)
10. Install on server or distribute through a cPanel Sync server.

Basic Sample Module

The package should be “cPanel::Category::Name”; (underscores are allowed and are displayed as spaces whenever appropriate)

Generally, none of the functions should be edited except as noted in the rest of the documentation

The only two required keys of the \$meta_info hash ref are used below:

```
package cPanel::CMS::Foo;

# always use these modules here:
use strict;
use warnings;

our $VERSION = '0.0.1'; # version of Foo.pm *not* the addon script's version

my $pkg = __PACKAGE__; # leave this as is

our $meta_info = { # see “$meta_info hash reference table” for details about this hashref
    'version' => '1.0',
    'description' => 'Foo is a tool to do Bar and Baz',
};

#### action functions ##
sub install { shift->stdinstall(@_) }
sub upgrade { shift->stdupgrade(@_) }
sub uninstall { shift->stduninstall(@_) }

#### non action functions ##
sub installform { print shift->{installform} }
sub upgradeform { print shift->{upgradeform} }
sub uninstallform { print shift->{uninstallform} }

#### addon specific special functions ##
1;
```

Creating a cPAddon Upgrade

An upgrade is a new version of a script. This new version should be created similar to the original cPAddon version of your script. The only differences should be the differences in your code for the script.

Now that Foo 1.2 has come out we simply:

1. Create Foo 1.2 just like we would a new cPAddon
2. Make Foo/upgrade/ if it does not exist already
3. Make Foo/upgrade/1.0_1.2/ (“old version”, underscore, “new version”)

This directory can have the following files (case sensitive)

* diff

This is the only required file and it is simply a patch from the command: `diff -ruN Foo_1.0.work/ Foo_1.2.work/`

Note that the directory needs to be the path you actually put in the tar ball. So if you made the tar ball of Foo_1.0/public_html/ then you will need to do the diff on Foo_1.0/public_html.

* mysql

Should contain any SQL needed to upgrade, you should use variable like Foo/1.2.mysql

* chmod

If diff creates files and you need to chmod them, list them here. Format is “700 file” per line (IE “mode file” where file is relative to install directory)

* remove

This should list files that are in the old version but not in the new version, one per line, that need to be deleted.

4. Remove Foo/1.0.*

There is only the need to create an upgrade from the previous version to the latest. So if 1.4 came out we'd make upgrade/1.2_1.4/ and any 1.0 installs would be upgraded to 1.4 by way of it following the upgrade path of 1.0 to 1.2 then 1.2 to 1.4

\$meta_info Hash Reference Table

Key	Value type	Value example	Details
adopt_url	String	http://addon.com/addition-funtionality-addon-modules/	Url to script's functionality expanding module.
admin_email	Boolean	1	Turns the email field on or off. If on it is available for config or mysql as [% email %]
admin_user_pass_length	Numeric	10	Minimum required length for [% username %] and [% password %]
admin_user_pass_length_max	Numeric	16	Maximum allowed length of [% username %] and [% password %]
adminarea_name	String	Foo Goods	.htaccess protected area name
adminarea_path	String	admin/	.htaccess protected path (protected with 'adminuser_pass' values), relative to the install directory
adminuser_pass	Boolean	1	Turns on or off the username and password fields. If on it is available for config or mysql as [% username %] and [% password %] respectively.
changelog_url	String	http://addon.com/changelog.txt	Url to script's changelog
chgrp	Boolean		Will recursively change the group of all install files to the install user's group.
chmod	Hash Ref	<pre>'chmod' => { '0600' => ['foo.conf'], },</pre>	each key is a 4 digit mode and the value is an array ref of files to chmod to the key relative to tarball insides
chmod_order	Array Ref		Add the keys in the chmod Hash Ref. These should be added in the order you want them applied.
chmod_recursively	Boolean		Will make the chmod action in chmod recursive.
config_files	Array ref	<pre>'config_files' => [qw(foo.conf admin/foo.conf)],</pre>	List of files that need processed as config files (IE have the [% variables %] expanded)
cron	String	1 * * * * [% installdir %]/foo 2&>1\n1 * * * * /whatever	crontab entry (including newlines !!) It is processed for the config variables
crypt_salt	String	xY	Specific salt to use when crypting the password
description	String	This is a widget for foo	A sentence or two that is a text description of what the script is.
documentation_url	String	http://addon.com/docs/	Url to script's documentation
installdir	String	myfoo	Initial value of the installation directory field
lisc_disc	Array ref	<pre>lisc_disc => ['40% off', 'cPanel42'],</pre>	The first item is the discount string and the second is a discount code to use when purchasing from the vendor
lisc_info	String	This product requires you buy a license from the vendor and enter the key after its installed the first time you log into the admin area.	Explanation of the vendor's license paradigm
lisc_url	String	http://vendor.com/buy_lisc.pl	Url to purchase a license
maintainer_url	String	http://addon.com/addon_contact.pl	Url to maintainer contact

mysql	Array ref	<code>'mysql' => ['foo'],</code>	Generally there's only one value for this array. It is a list of short <code>^\w+\$</code> strings that are used to reference the databases(es) used by the cPAddons.
nomovecopy	Boolean	1	Turns copy and move functionality on or off
packager_name	String	My Addon Co.	Name of packager
perl_module	Hash ref	<code>'perl_module' => { 'Mod::You::Need' => '1.2', 'Other::Mod' => '', },</code>	Warns if any of the required Perl modules need installed. Each key is the name space of the module and the value is either the required minimum version or an empty string (meaning *any* version)
require_suexec	Boolean	1	Only allow installation if suexec is enabled.
security	String	Do not allow users to post HTML when posting	Text description of security issues good or bad with the script.
setphpsuexecvar	Boolean	1	Turns on the check to see if phpsuexec is enabled or not
specialfunctions	Hash ref	<code>'specialfunctions' => { 'checkinstallsforfleas' => { 'code' => \&checkinstallsforfleas, 'name' => 'Check Installs for fleas', }, },</code> 'code' and 'name' are required for each special functions hashref key	Add any additional functionality to managing the installs in cPAddons sub checkinstallsforfleas {} should be put after the "#### addon specific special functions ##" in our example module above
support_url	String	<code>http://addon.com/support/</code>	Url to vendor's support
table_prefix	String	foo	If its specified, turns on table prefix input field, available via [% table_prefix %].Its also used as the initial value of that field.
vendor_license	Depends	Depends	The license key for your addon, if needed. See Licensing Your Addons
version	String	1.0.42-beta	Version of script that this Module installs
warnphpsuexec_change	String	PHPSuExec functionality has changed since this install was done. Use the fix suexec status below to resolve any PHPSuExec issues	What to say when phpsuexec status has changed (if setphpsuexecvar was in use)
warnsuexec_change	String	warning, steps to take if necessary	Like 'warnphpsuexec_change' but for suexec
website	String	<code>http://addon.com/</code>	Url to vendor's website
whm_addon	Array ref	<code>['tomcat']</code>	List of WHM Addon Modules that are required for installation. If your addon requires a system component to be installed it must be installed as a WHM Addon Module. For more info click here.

'1.0.42-beta' -> {public_html_install_files}	Array ref	['admin.html', 'README', 'other_file_name']	This key designates what files are associated with the installer. If you add these, they will be removed when the addon is uninstalled.
'1.0.42-beta' -> {public_html_install_dirs }	Array ref	['admin/foo', 'admin'],	This way if a user has there own files in those directories the directories will not be deleted even though the installed files will be gone (from *_files above - list directories in "cascading order" or rmdir() will be unable to do its thing
'1.0.42-beta' -> {public_html_install_unknown }	Array ref	['uploads'],	for directories that will have unknown contents (IE files uploaded to it) or that you otherwise don't want to specify every single file in above - these directories are rm -rf'ed :)
'1.0.42-beta' -> {public_html_install_note }	String	The files will also be removed when uninstalled so be careful with common named files or directories!	This note should warn users if you are using common names and directory names for your install. I.E. if you use index.html and add that to 'public_html_install_files' any file named index.html will be removed from the install directory.

Custom Install Fields

If you need custom installation fields other than those included in the basic cPAddons install, those are available as well. For example, if you need a secondary contact email, you may need a custom field. If you need specific info for a proper installation you can use the following additional keys for the \$meta_info hash ref:

install_fields

“install_fields” will build your install form to make sure the input you need exists. Here, you can add your extra fields for text, radio buttons, etc.

This hashref is used to build the input fields, here are some guidelines:

- * Do not use addon, email, auser, apass, installdir, or action as your keys they are reserved and will be ignored.
- * Each “type” below must use the value as is (IE if its a hashref below it always has to be a hashref , if its a string its always got to be a string.
- * The key is the name attribute of the form field. You can use them in your config and sql files with [% input_NAME %] where “NAME” is the key.
- * If multiple entires exist (IE multiple select or checkbox they will be [% input_NAME %] [% input_NAME-0 %] [% input_NAME-1 %] etc
- * You are responsible for verifying the input is correct and doing any quoting or erroring out *before* the install function is run.
 - o If a module is being moderated and you die/quit/exit in your install function (IE saying “invalid motto please re enter your motto”) then the moderation approval will need redone so use caution!!! or better yet - use install_fields_hook => :)

Here is a sample hashref that contains each type of install field:

```
'install_fields' => {
  'test_text' => {
    'label' => 'Text',
    'value' => 'text value',
    'attr' => 'size="32"',
    'type' => 'text',
  },
  'test_hidden' => {
    'label' => 'Hidden',
    'value' => 'Hidden value',
    'attr' => 'attr="2"',
    'type' => 'hidden',
  },
  'test_password' => {
    'label' => 'Password',
    'value' => 'password value',
    'attr' => 'size="10"',
    'type' => 'password'
  },
  'test_textarea' => {
    'label' => 'Text Area',
    'value' => 'textarea text',
    'attr' => 'cols="51"',
    'type' => 'textarea',
  },
  'test_select' => {
    'label' => 'Select',
    'value' => { foo => 'FOO', bar => 'Bar' },
    'attr' => '',
    'type' => 'select',
    'defvalue' => 'bar',
  },
  'test_radio' => {
    'label' => 'Radio',
    'value' => { 1 => 'One', 2 => 'Two', 3 => 'Three' },
    'attr' => '',
    'type' => 'radio',
    'defvalue' => '2',
  },
  'test_multi' => {
    'label' => 'Multi',
    'value' => { 4 => 'Four', 5 => 'Five', 6 => 'Six', 7 => 'Seven' },
    'attr' => 'size="2"',
    'type' => 'multi',
    'defvalue' => [5,7],
  },
  'test_checkbox' => {
    'label' => 'Checkbox',
    'value' => { 8 => 'Eight', 9 => 'Nine', 10 => 'Ten', 11 => 'Eleven' },
    'attr' => '',
    'type' => 'checkbox',
    'defvalue' => [8,11],
  },
}
```

install_fields_hook

You should always use `install_fields_hook` when using `install_fields`. The `install_fields_hook` checks input from the form generated using `install_fields`. It should be modified to make sure input is parsed properly and will not corrupt your configuration or MySQL files. Failure to properly evaluate and escape this input may result in a broken installation.

If you need to modify any input, you will need to modify `$obj->{'input_whatever'}` since the object key is used as a config variable. Changing `$input_hr->{'whatever'}` will have no effect on what gets put into the install.

```
'install_fields_hook' => sub {
  my ($input_hr, $error_hr, $obj) = @_ ;
  # escape single quotes so it doesn't break the
  # single quoted value in the config file (= '[% install_foo %]';)
  $obj->{'input_foo'} =~ s/{'}{\'}g;

  # check $input_hr and if there are problems set the error like so:
  print "Checking your laces...<br />\n";
  ${$error_ref} = "Your shoes are untied, try again"
    if !_laces_are_tied_tight($input_hr->{'shoes'});
},
```

Config File Variables

In a config file (or mysql or other value documented as using these variables) there are many variables that are available. If you need custom configuration variables for your script's installer, you will need to define those variables both inside your install / upgrade functions and inside your configuration file or any other location using these variables. The basic paradigm is [% variable_name %] (left square bracket, percent sign, space, \w+ string name, space, percent sign, right square bracket) You can create customized ones by setting it in the object inside the given function.

```
sub install {  
  my $obj = shift;  
  $obj->{'vendor_special_value'} = _get_random_vendor_special_value();  
  $obj->stdinstall(@_);  
}
```

Then you can get it into the config file by having this in your config file:

```
$special_value = "[% vendor_special_value %]",
```

The following variables are used in the default config file:

```
$VAR1 = {
    'mysql.$meta_info->{mysql}[0].sqladb' => 'database_name', #MySQL database name
    'installed_on_domain' => 'domain.com', #Domain associated with addon
    'mysql_user_post' => 'uniqueid', #Unique id after database name i.e database_name1
    'workinginstall' => '', #Internal use only
    'hostname' => 'server.domain.com', #Hostname of the server
    'email' => 'user@domain.com', #Contact email from install form
    'mysql.$meta_info->{mysql}[0].sqluser' => 'user_uniqueid', #mysql user_uniqueid
    'url_to_install' => 'http://domain.com/dir/', #URL the addon is installed at
    'password' => 'abcdefghijkl', #Password from install form
    'unix_time' => 1141098572, #Server Time at Install
    'addon' => 'Vendor::Category::Name', #Addon's name space
    'mysql_pass' => 'abcdefghijklm', #Password for MySQL database user
    'user' => 'aoz', #Database user
    'salt' => 'abbccdde', #Salt for password encryption
    'lang' => 'english', #Language used for install
    'installed_1.5.2' => 1141098575, #Timestamp of when the version is installed
    'mysql.$meta_info->{mysql}[0].sqlpass' => 'abcdefghijkl', #MySQL database password
    'username' => 'abbccdde', #Username for install
    'mysql.$meta_info->{mysql}[0].mysql-version' => '4.1', #MySQL version on server
    '$meta_info->{mysql}[1]' => 'user_uniqueid', #Mysql user_uniqueid
    'table_prefix' => 'pfix', #MySQL Table Prefix (keep this under 4 chars [^\w+$ string])
    'installpath' => 'dir', #Directory where addon will be installed
    'registry' => 'Vendor::Category::Name.0', #~/cpaddons/file_where_config_is_stored
    'public_html' => '/home/user/public_html', #Location of the user's document root
    'domain' => 'domain.com', #Domain the addon is installed for
    'adminarea_name' => undef, #Name of admin area for addon
    'addon_path' => 'Vendor/Category/Name', #Path of install files on server
    'password_md5_hex' => '6R54Ty3', #Hexadecimal MD5 sum of password in install form
    #Note: This is the most commonly used MD5 sum)
    'password_md5' => 'g5;$3e', #MD5 sum of password from install form (binary MD5)
    'url_to_install_admin' => http://domain.com/dir, #URL to install admin interface to
    'version' => '1.5.2', #Version of addon installed
    'installdir' => '/home/user/public_html/dir', #Install path on the server
    'mysql.$meta_info->{mysql}[0]._post' => 'uniqueid', #Uniqueid after database name
    'mysql_user' => 'domainuser_dbuser', #MySQL Database user
    'autoupdate' => 0, #Autoupdate addon installs (0=no 1=yes)
    'suexec' => 1, #Use suexec (0=no 1=yes)
    'homedir' => '/home/user', #Domain user's home directory location
    'password_crypt' => 'dfiF72Po.urHc', #[% password %] crpyted using [% salt %]
    'gmt_hour_offset' => '-6', #Hours away from GMT the server is
    'mysql_version' => '4.1', #Version of MySQL on server
    'password_md5_base64' => 'gZFn5TS30qjWeXKcBQpgjA', #Base64 md5 sum of [ % password % ] (or password
    #from install form)
    'url_to_install_without_trailing_slash' => 'http://domain.com/dir', #Directory to install to without trailing slash
    'no_protocol_url_to_install' => 'domain.com/dir/' #Directory to install to without http://
    'no_protocol_url_to_install_without_trailing_slash' => 'domain.com/dir' #Directory to install to without http:// or trailing slash
    'mysql.$meta_info->{mysql}[0].sqlhost' => 'hostname', #Server the MySQL database is on (usually localhost)
    'phpsuexec' => '' #Use phpsuexec (0=no 1=yes)
};
```

Invisible cPAddon Scripts

If you create a cPAddon that is in some way associated with another cPAddon, or should otherwise not be “advertised” to all users, you can make it not show up in cPanel until at least one is installed on the user’s site.

This is useful only if you make a link to it in the description of the regular cPAddon it is associated with or otherwise reference it for whoever might need to use it. (say in a document to your server techs or in an email to a customer informing them of a special option they have that is otherwise hidden).

For example, if you created a patched version of phpBB that adds some special functionality such as a new theme, you may only want to make this patched version available if the user is already using phpBB.

You can make “invisible” addons by prefacing the name with an underscore:

```
3rdParty::Whatever::_phpBB_with_foo_patch
```

You might find these useful for:

- * Temporary installations (maybe a troubleshooting or diagnostic system for a tech to use to resolve a problem of some sort or use for testing)
- * cPAddons that might interact with each other in some way (IE one that installs drop in modules to an existing installation)
- * Alternate versions (Lite, Pro, Ecommerce addition, Romanian Centric)
- * Scripts that an admin may want readily available for use but does not want to advertise as available
- * Specialized scripts that most people would have not use for and perhaps find confusing (Development, Astrophysics, Quantum Physics, DNA Sequencing, etc)

Here is an example of a cPAddon that references 2 “invisible” related cPAddons:

```
package Vendor::CMS::FooPro

use strict;
use warnings;

our $VERSION = '1.0.1';

my $pkg = __PACKAGE__;

our $meta_info = { # see “$meta_info hash reference table” for details about this hashref
    'version' => '1.0',
    'description' => q(This are also two specialized versions of FooPro with additional functionality. You can install these )
        . qq(themes by clicking here: [
```

Addon Aliases

If an addon could fit into more than one category simply create an alias. For example, if a Content Management System could also be a photo gallery, a wiki, etc. This can be accomplished like so:

```
package Vendor::CMS::My_Content_Management_System;

use strict;
use warnings;

our $VERSION = 0.1;

my $pkg = __PACKAGE__;

our $meta_info = {

    # required, name space for what its an alias for
    'alias' => 'Vendor::Blog_Engine::My_Content_Management_System',

    # if description is not specified it simply says "This is an alias for: XYZ"
    'description' => 'This is a powerful blog engine which also is a CMS',
    'security_rank' => 10, # should be the same or higher that what its an
                          # alias for since it may error out if its below
                          # your security rank limit
};
1;
```

Addons that Require Licenses

If your script requires a commercial license you have 3 options:

1. Handle the license requirements and entry from the installed addon
2. Use the `$meta_info` key “`vendor_license`” to install regardless of if a valid license was entered (IE your addon script will just detect an invalid license and handle as you wish)
3. Use the `$meta_info` key “`vendor_license`” to install *only* if the license entered was valid

The first step is to create an ‘`install_field`’ key called ‘`vendor_license`’ as well as a simple check in ‘`install_field_hook`’.

After that:

To simply allow them to enter a `^\w+$` license key and put it in [% `vendor_license` %]:

```
‘vendor_license’ => {},
```

To allow them to enter a license key of any format you wish and put it in [% `vendor_license` %]:

```
‘vendor_license’ => {  
  ‘string_is_ok’ => sub {  
    my($lisc_string) = @_; # 32 digits  
    return 1 if length $lisc_string == 32 && $lisc_string =~ m/^\d+$/; # ^\d{32}$  
    return 0;  
  },  
},
```

The following will allow the customer to enter anything into the field and have a script verify the license key is valid or else not install/upgrade it:

```
‘vendor_license’ => {  
  ‘verify_url’ => ‘http://foo.bar.baz/lisc.pl’, # ?lisc=license_string_from_input_here  
  ‘url_says_its_ok’ => sub {  
    my ($url_result) = @_; # it is empty if url failed  
    return 1 if $url_result =~ m/Vendor says Valid/;  
    return 0;  
  },  
},
```

Distributing Your Addons

If you wish for your addon scripts to be able to be installed through WebHost Manager, you will need to be able to distribute them to your customers. To do this, you will need to set up a cPanel Sync Server. See the 'Creating Your cPanel Sync Server' for more information.

If you choose not to set up a cPanel Sync Server, each addon script will need to be installed manually on each cPanel server.

Creating a Vendor Name

If you have one or more cPAddons you wish to make available for installation via cPanel you should set yourself up as a "Vendor". Once you have a vendor setup, for example: "TMBG_Inc", you can create as many addons as you like under the TMBG_Inc::Category::Name name space using the directions in Creating a New cPAddon Script. The first step is to pick a short descriptive Vendor name that identifies the individual or organization behind them. It must be all numbers, letters, and underscores. To create a Vendor name, use a custom name when creating your addon script. For example: package Vendor::Category::Name; or John::CMS::Johns_Content_System; For more information on creating an addon script, see Basic Sample Module. Distributing your software People will be able to add you as vendor to use in WHM by entering your "cPAddon Vendor Info Url" under Install cPAddon Scripts.

The url you give out to use in WHM as your "cPAddon Vendor Info url" should be the url of this script:

```
#!/usr/bin/perl

use strict;
use warnings;

use CGI qw(header param);

print header('text/plain');

my $data = param('cphost') ? 'your.cpsync.host.here'
    : param('cphuri') ? '/cpaddons/path/on/cphost/here'
    : param('palmd5') ? 'MD5_Sum_of_your_/cPAddonsMd5/Vendor.pm_here'
    : 'Your_Vendor_Name_Here'; # must be ^\w+$
print $data;
```

You will need to change the 4 pieces of data to reflect your hostname, path of the addon, md5 sum of your addon package and your vendor name.

In the example on the previous page, the url of the base cPAddons directory would be:
<http://your.cpsync.host.here/cpaddons/path/on/cphost/here/>

The URL will need to point to a server using cPanel Sync where all files for your vendorship and cPAddons will be located.

Those files would be:

* cPAddonsAvailable/Vendor.pm

```
package cPAddonsAvailable::Vendor;
our %list = (
    'Vendor::Category::Name' => {
        'VERSION'    => '0.0.1', # $Vendor::Category::Name::VERSION
        'version'    => '1.0-beta', # script's version, same as $meta_info->{'version'}
    },
    # repeat for each cPAddon you want to distribute
);
1;
```

* cPAddonsMD5/Vendor.pm (update the md5 sum of this file in “cPAddon Vendor Info Url” when you edit it)

```
package cPAddonsMD5::Vendor;
our %cpaddons = (
    'Vendor::Support::Mars_Rover' => {
        'md5' => '024994ae44c700902ced23c777cf25d2', # md5 of .pm file
        'desc' => 'This script is a Perl/MySQL based Mars Rover Support system',
    },
    # repeat for each cPAddon you want to distribute
);
1;
```

* Vendor/*/*

These are the actual cPAddons you create: Vendor/Catagory/Foo/* Vendor/Catagory/Foo.pm

Installing Addons Manually

If you choose not to use a cPanel Sync Server for your addons, they will need to be installed manually on each server the addon will be used on.

To install your addon on the server:

Add your addon files to `/usr/local/cpanel/cpaddons/Vendor/Category/`

This includes your `addon.pm` and your addon directory

Your addons will only show up if the Vendor is in the `%vend` hash of `/usr/local/cpanel/cpaddons/cPAddonsConf.pm`

NOTE: If you choose this method, you will need to install your addon in an existing Vendor namespace. If you install your addon into the cPanel namespace, your addon will show as not trusted.

Licensing your Addons

If your addon requires a commercial license you have 3 options:

1. Handle the license requirements and entry from the installed addon only. (100% independent of the cPanel system)
2. Use the `$meta_infoLicensing Your Addons` key “vendor_license” to install regardless of if a valid license was entered (IE your addon script will just detect an invalid license and handle as you wish)
3. Use the `$meta_info` key “vendor_license” to install only if the license entered was valid as determined by your license server on the fly.

The first step is to create an ‘install_field’ key called ‘vendor_license’ as well as a simple check for the license in ‘install_field_hook’.

After that:

To simply allow them to enter a `^\w+$` license key and put it in `[% vendor_license %]`:

```
‘vendor_license’ => {},
```

To allow them to enter a license key of any format you wish and put it in `[% vendor_license %]`:

```
‘vendor_license’ => {
  ‘bad_string_error’ => ‘Needs to be 32 digits’, # optional description of problem with
    the input as determined by the code ref below
  ‘string_is_ok’ => sub {
    my($lisc_string) = @_; # 32 digits
    return 1 if length $lisc_string == 32 && $lisc_string =~ m/^\d+$/; # ^\d{32}$
    return 0;
  },
},
```

To allow them to enter anything they want and have a script verify its valid or else not install/upgrade it:

```
‘vendor_license’ => {
  ‘use_url_as_error’ => 1, # optional, turns on or off using the result of the url in the same way
    #as ‘bad_string_error’ above
    # it would be best to just return short text string
  ‘verify_url’ => ‘http://foo.bar.baz/lisc.pl’, # ?user=cpanel_user&lisc=license_string_
    # from_input_here
  ‘url_says_its_ok’ => sub {
    my ($url_result) = @_; # it is empty if url failed
    return 1 if $url_result =~ m/Vendor says Valid/;
    return 0;
  },
},
```

Creating Your cPanel Sync Server

If you wish to distribute your addons through a cPanel Sync Server, you will need to configure the server on which your addons reside to be a cPanel Sync Server.

You can find Utilites and information for creating cpanelsync aware directories at <http://search.cpan.org/perldoc?cPanel::SyncUtil>

Please read the documentation (and see the fully working sample scripts in the distribution) at that url to learn more about using this utility to create a cPanel Sync Server.

There is a script called `cpanelsync_build_cpaddons_dir` in the distribution's `/scripts` directory that you can use to build your `cpanelsync` directory for your `cPAddons`. You will need to copy it to your server to a location that is not publicly accessible and execute it on your server as root. Then, follow its prompts to see what data you need to supply it so that it can build what you need.

For example: `./cpanelsync_build_cpaddons_dir -d /home/user/public_html/cpaddons/path/on/cphost/here -u user -a bz2 -v myvendor`

This will make `/home/user/public_html/cpaddons/path/on/cphost/here` a cPanel Sync directory with all files owned by `user` and from the vendor 'myvendor'. The `-a bz2` is required for all addons. .

For more information on the script, execute `./cpanelsync_build_cpaddons_dir` with no options.

cPanel::SyncUtil

Module Version: 0.0.1

NAME-

cPanel::SyncUtil - Perl extension for creating utilities that work with cpanelsync aware directories

SYNOPSIS:

```
use cPanel::SyncUtil;
```

DESCRIPTION:

These utility functions can be used to in scripts that create and work with cpanelsync environments.

EXAMPLE:

See scripts/cpanelsync_build for a working example that can be used to build cPanel's cPAddon Vendor cpanel-sync directory for your website.

EXPORT:

None by default, all functions are exportable if you wish:

```
use cPanel::SyncUtil qw(_raw_dir);  
use cPanel::SyncUtil qw(:all);
```

FUNCTIONS:

`_chown_pwd_recursively`

Takes as its first argument a user that matches `^\w+$` (and optionally a group as its second argument, also matching `^\w+$`) and recursively chown's the current working directory to the given user (and group if given).

Currently the return value is from a `system()` call to `chown`.

`_safe_cpsync_dir`

Returns true if the given argument is a directory that it is safe to be cpanelsync'ified.

See the simple, scripts/cpanelsync_build_dir script for example useage while recursing directories.

`_raw_dir`

This function makes the .tar and .bz2 version of the file system.

Its arguments are the following:

```
_raw_dir($base, $archive, $verbose, @files);
```

`$base` and `$archive` are the only required arguments.

`$archive` is a directory in `$base`.

It will `chdir` in `$base` and then process the directory `$archive`

If `$verbose` is true, output will be verbose.

If `@files` is specified each item in it is also processed.

Each item in `@files` must be a file (`-f`) in `$base/$archive`.

If it returns false the error is in `$!`

```
_raw_dir($base, $archive, $verbose, @files)  
or die "_raw_dir($base, $archive, $verbose, @files) failed: $!";
```

It's very important to check the return value because if it failed it's possible you will not be in the directory you think and then subsequent file operations will either fail or not work like you expect. Plus if it returned false then there is either a file system problem or the input to the function is not valid. In other words, if it fails you need to resolve the problem before continuing so `die()`ing is a good idea generally.

`_sync_touchlock_pwd` is then run on `$base/$archive` so that it's now a `cpanelsync` directory

`_get_opts_hash`

Shortcut to get a hash (in array context) or hash ref (in scalar context) of the script using this module's command line options.

Takes the same exact input as `Getopt::Std::getopts()`

`_sync_touchlock_pwd`

Creates the `.cpanelsync` file (and its `.bz2` version) and `.cpanelsync.lock` for the current working directory

`_read_dir`

Shortcut to `File::Slurp`'s `read_dir`

`_write_file`

Shortcut to `File::Slurp`'s `write_file`

`_lock`

Locks the given directories.

```
_lock(qw(foo bar baz));
```

`_unlock`

Unlocks the given directories.

```
_unlock(qw(foo bar baz));
```

SEE ALSO:

cPanel, <http://www.cpanel.net>

TODO:

replace `system()` calls with perl versions.

anything mentioned in the source

AUTHOR:

Daniel Muey, http://drmuey.com/cpan_contact.pl

COPYRIGHT AND LICENSE:

Copyright (C) 2006 cPanel, Inc.

This library is free software; you can redistribute it and/or modify it under the same terms as Perl itself, either Perl version 5.8.6 or, at your option, any later version of Perl 5 you may have available.