Tutorial - Create a Standardized Hook

Introduction

This tutorial adds custom hooks to preserve a customized configuration file. The cPanel & WHM update process (`upcp`) sometimes overwrites customized files with the cPanel-provided default versions. If you experience this problem, use our Standardized Hooks system to preserve your customizations.

To solve this problem, this tutorial creates the following hooks:

- A `pre`-stage hook that runs immediately before the `upcp` script to copy the customized Horde configuration file (`/usr/local/cpanel/base/horde/turba/config/backends.php`) to the `backends.php.temp` file.
- A `post`-stage hook that runs immediately after the `upcp` script to copy the `backends.php.temp` file back into the `backends.php` file, and then deletes the `backends.php.temp` file.

> **Note:**
> This tutorial creates a Perl module, but you can also write hook action code in PHP or any other preferred programming language. For more information, read our Hook Action Code documentation.

Create and register hooks

Create a file to contain your hook action code.

Create a new Perl file to contain your hook action code.

- The filename that you use **must** meet the internal Perl interpreter's requirements for module names.
- This tutorial uses the `MyHook.pm` file.

Package the module.

This declaration instructs Perl to treat all of the file's functions as part of the `MyHook` module. For more information, read perldoc.perl.org's package documentation.

```
# Package this module.
package MyHook;
```

Set the strict pragma and use warnings.

These declarations instruct Perl to return errors if the file contains potentially-unsafe code.

> **Note:**
> You can omit the `warnings` declaration in production code, but we **strongly** recommend that you use it during development.

In this tutorial

Related documentation

- Perl Module Installation
- Scripts and Scripting Languages FAQ
- Install a Perl Module
- Module Installers
- Perl Modules

```
# Return errors if Perl experiences
problems.
use strict;
use warnings;
```

Use additional modules.

These declarations instruct Perl to use the following additional modules:

- `Cpanel::Logger` — This module uses cPanel & WHM's logging functionality to write messages to log files.
- `File::Copy` — This core Perl module includes methods to manipulate files.
- `JSON` — This module properly encodes and decodes JSON.

```
# Use cPanel's error logging module.
use Cpanel::Logger;

# Use the core Perl module with
file-copying functionality.
use File::Copy;

# Properly decode JSON.
use JSON;
```

Instantiate a new `Cpanel::Logger` object.

Use the `Cpanel::Logger` module's `new()` method to instantiate a new logging object.

```
# Instantiate the cPanel logging object.
my $logger = Cpanel::Logger->new();
```

Note:
By default, the `new()` method creates an object that logs to the `usr/local/cpanel/logs/error_log` file.

- You can also specify a custom log location.
- For more information, read our Guide to Standardized Hooks - Custom Event Handlers documentation.

Create a `describe()` method to embed hook attributes in your code.

Create a `describe()` method that contains the attributes for all of the hooks in the code. This step is optional, but it allows you to simplify the way in which you call the b `in/manage_hooks` utility when you register the hooks.

For this tutorial, the `describe()` method includes the attributes for the following hooks:

- The first hook executes the `MyHook` module's `copyfile` subroutine during the `pre` stage of the `System` category's `upcp` event.
- The second hook executes the `MyHook` module's `replacefile` subroutine during the `post` stage of the `System` category's `upcp` event.

For more information, read our Guide to Standardized Hooks - The manage_hooks Utility documentation.

```
# Embed hook attributes alongside the
action code.
sub describe {
    my $hooks = [
        {
            'category' => 'System',
            'event'    => 'upcp',
            'stage'    => 'pre',
            'hook'     =>
'MyHook::copyfile',
            'exectype' => 'module',
        },{
            'category' => 'System',
            'event'    => 'upcp',
            'stage'    => 'post',
            'hook'     =>
'MyHook::replacefile',
            'exectype' => 'module',
        }
    ];
    return $hooks;
}
```

Specify the files to manipulate.

Create variables that contain the absolute paths for the configuration file that you wish to preserve and the temporary file that you wish to create in order to preserve it.

```
# Set the file to preserve and the temp
file location.
my $preserve =
"/usr/local/cpanel/base/horde/turba/con
fig/backends.php";
my $temp =
"/usr/local/cpanel/base/horde/turba/con
fig/backends.php.temp";
```

Note:
The temporary file's location is arbitrary.

Create the `pre` hook's subroutine.

The system will run the `copyfile` subroutine immediately before the `upcp` process.

- Line 49 reads the entire `@_` stream until end-of-line (EOL).

  Important:
  - Every hook subroutine **must** perform this action.
  - Because this code uses the `JSON` module, it treats this data as a JSON-encoded data structure. After the system decodes the JSON string, the native data structure becomes a hash.

- Lines 52 adds a logging action to the subroutine. This step is optional, but we **strongly** recommend that you include it for debugging purposes.
- Line 55 uses the `File::Copy` module's `copy()` function to copy the contents of the `$preserve` file into the `$temp` file.
  - If the `$temp` file does not already exist, the system will automatically create it.
  - If the copy fails, the system will log a failure message that includes any error messages.

    Warning:
    Failure does **not** block the `upcp` event.

```
# Before upcp, copy the file to the temp
file.
sub copyfile {
    # Get the data that the system
passes to the hook.
    my ( $context, $data ) = @_;

    # Add a log entry for debugging.
    $logger->info("***** Copy my file
before upcp *****");

    # Copy my file.
    copy($preserve,$temp) or die "Copy
failed: $!";
};
```

Note:
Make certain that you use the same subroutine names that you specified in the `describe()` method in step 6.

Create the `post` hook's subroutine.

The system will run the `replacefile` subroutine immediately after the `upcp` process .

- Line 61 reads the entire `@_` stream until end-of-line (EOL).

  Important:
  - Every hook subroutine **must** perform this action.
  - Because this code uses the `JSON` module, it treats this data as a JSON-encoded data structure. After the system decodes the JSON string, the native data structure becomes a hash.

- Lines 64 and 70 add logging actions to the subroutine. This step is optional, but we **strongly** recommend that you include it for debugging purposes.
- Line 67 uses the `File::Copy` module's `copy()` function to copy the contents of the `$temp` file back into the `$preserve` file.
  - If the `$preserve` file no longer exists, the system will automatically create it.
  - If the copy fails, the system will log a failure message that includes any error messages.
- Line 73 deletes the `$temp` file.
  - This step is optional, but ensures that your system does not retain unnecessary files.
  - If the deletion fails, the system will log a failure message that includes any error messages.

```
# After upcp, copy the file back into
place.
sub replacefile {
    # Get the data that the system
passes to the hook.
    my ( $context, $data ) = @_;

    # Add a log entry for debugging.
    $logger->info("***** Replace the
temp file *****");

    # Replace my file.
    copy($temp,$preserve) or die
"Replacement failed: $!";

    # Add a log entry for debugging.
    $logger->info("***** Delete the temp
file *****");

    # Delete the temp file to keep
cruft off my system.
    unlink($temp) or die "Cruft removal
failed: $!";
};
```

Note:
Make certain that you use the same subroutine names that you specified in the `describe()` method in step 6.

Save the file.

Save your hook action code to one of the following locations on your cPanel & WHM server:

- `/usr/local/cpanel`
- `/usr/local/cpanel/3rdparty/perl/514/lib64/perl5/cpanel_lib/x86_64-linux-64int`
- `/usr/local/cpanel/3rdparty/perl/514/lib64/perl5/cpanel_lib`
- `/usr/local/cpanel/3rdparty/perl/514/lib64/perl5/5.14.4/x86_64-linux-64int`
- `/usr/local/cpanel/3rdparty/perl/514/lib64/perl5/5.14.4`
- `/opt/cpanel/perl5/514/site_lib/x86_64-linux-64int`
- `/opt/cpanel/perl5/514/site_lib`
- `/var/cpanel/perl5/lib`
- `/var/cpanel/perl5/lib`

Important:
You **must** save hook action code in one of these locations, or you will receive an `@INC` error when you attempt to register the hook.

Your final file should appear similar to the following example:

Click to view complete Perl code...

```
#!/usr/bin/perl

# Package this module.
package MyHook;

# Return errors if Perl experiences
problems.
use strict;
use warnings;

# Use cPanel's error logging module.
use Cpanel::Logger;

# Use the core Perl module with
file-copying functionality.
use File::Copy;

# Properly decode JSON.
use JSON;

# Instantiate the cPanel logging
object.
my $logger = Cpanel::Logger->new();

# Embed hook attributes alongside the
action code.
sub describe {
    my $hooks = [
```

```
        {
            'category' => 'System',
            'event'    => 'upcp',
            'stage'    => 'pre',
            'hook'     =>
'MyHook::copyfile',
            'exectype' => 'module',
        },{
            'category' => 'System',
            'event'    => 'upcp',
            'stage'    => 'post',
            'hook'     =>
'MyHook::replacefile',
            'exectype' => 'module',
        }
    ];
    return $hooks;
}

# Set the file to preserve and the
temp file location.
my $preserve =
"/usr/local/cpanel/base/horde/turba/c
onfig/backends.php";
my $temp =
"/usr/local/cpanel/base/horde/turba/c
onfig/backends.php.temp";

# Before upcp, copy the file to the
temp file.
sub copyfile {
    # Get the data that the system
passes to the hook.
    my ( $context, $data ) = @_;

    # Add a log entry for debugging.
    $logger->info("***** Copy my file
before upcp *****");

    # Copy my file.
    copy($preserve,$temp) or die
"Copy failed: $!";
};

# After upcp, copy the file back into
place.
sub replacefile {
    # Get the data that the system
passes to the hook.
    my ( $context, $data ) = @_;

    # Add a log entry for debugging.
    $logger->info("***** Replace the
```

```
temp file *****");

    # Replace my file.
    copy($temp,$preserve) or die
"Replacement failed: $!";

    # Add a log entry for debugging.
    $logger->info("***** Delete the
temp file *****");

    # Delete the temp file to keep
cruft off my system.
    unlink($temp) or die "Cruft
removal failed: $!";
```

```
};

1;
```

Register your hooks.

Run the following command as the `root` user to register the two new hooks:

```
bin/manage_hooks add module MyHook
```

If the system registered the hooks correctly, the script produces the following output:

```
Added hook for System::upcp to hooks
registry
Added hook for System::upcp to hooks
registry
```

Note:
If you do not include the `describe()` method in your custom code, you **must** include additional information when you run this script. For more information, read our Guide to Standardized Hooks - The manage_hooks Utility documentation.

Test your hooks.

To test `upcp` hooks, click *Click to Upgrade* in WHM's *Upgrade to Latest Version* interface (*WHM >> Home >> cPanel >> Upgrade to Latest Version*) or run the following command:

```
/usr/local/cpanel/scripts/upcp
```

After the `upcp` process finishes, the `/usr/local/cpanel/logs/error_log` file should contain the following entries:

```
[2015-06-16 11:08:29 -0500] info [hook]
***** Copy my file before upcp *****
[2015-06-16 11:08:29 -0500] info [hook]
***** Replace the temp file *****
[2015-06-16 11:08:29 -0500] info [hook]
***** Delete the temp file *****
```

If some or all of these entries do not appear in the error log, or if you see error messages in the same portion of the file, you may need to troubleshoot issues on your system.

Troubleshooting.

If you experience problems with hooks, the following troubleshooting methods may help you to find and fix the issue:

Check whether your hooks registered properly.

To view a list of registered hooks, run the following command:

```
/usr/local/cpanel/bin/manage_hooks list
```

If the hooks registered successfully, the output should resemble the following example:

```
System:
 upcp:
  stage: pre
  weight: 100
  id: lb6vAuWAZ9JNmzIIEsiLAeJw
  exectype: module
  hook: MyHook::copyfile
  --
  stage: post
  weight: 200
  id: NhhoUZ1GnGSfpi3dpcClKrnT
  exectype: module
  hook: MyHook::replacefile
```

Add more logging to your hook action code.

To cause your hook action code to log additional information, perform the following steps:

1. Add the following subroutine to your hook action code:

```
# Log extra info for debugging.
sub _more_logging {
    my ( $context, $data ) = @_;
    my $pretty_json =
JSON->new->pretty;
    $logger->info(
$pretty_json->encode($context) );
    $logger->info(
$pretty_json->encode($data) );
}
```

2. Add the following line of code to any points in your code that you believe may

cause the problems:

```
_more_logging( $context, $data );
```

This subroutine causes the function to log additional information about the hook event to the error log file, which appears similar to the following example:

```
[2015-06-16 11:08:29 -0500] info [hook]
{
   "stage" : "pre",
   "point" : "main",
   "category" : "System",
   "event" : "upcp"
}
```