Guide to Standardized Hooks - Custom Event Handlers

Introduction

> **Warning:**
>
> This method is **deprecated**. To convert custom event handlers to use the Standardized Hooks system, use the `API::Module::function` `Cpanel` event in your Hook Action Code.

Custom event handlers run before or after cPanel API functions. Each time that the system runs a cPanel API 1, cPanel API 2, or UAPI function, the `cpsrvd` daemon calls the `event` subroutine in the `CustomEventHandler.pm` file. In the `event` subroutine, you can create custom actions to run before or after an API function.

- The `/usr/local/cpanel/Cpanel/CustomEventHandler.pm` file **does not** exist by default. You **must** create it.
- cPanel & WHM includes an example in the `/usr/local/cpanel/Cpanel/CustomEventHandler.pm.sample` file.
- Debugging tools for custom event handlers are available.

Basic usage

> **Warning:**
>
> Any errors in the `CustomEventHandler.pm` file will cause the operation to fail. Test custom event handlers **thoroughly** before you attempt to use them on production servers.

```
package Cpanel::CustomEventHandler;

use strict;

use Cpanel::Logger ();

sub event {
    my ( $api_version, $type, $module, $func, $cfgref, $dataref
) = @_;
    if ($module eq 'ftp') {
        if ($func eq 'addftp') {
                # Do something for the addftp function.
        } elsif ($func eq 'delftp') {
        # Do something for the delftp function.
        }
    }
    return 1;
}
1;
```

Package the module

```
package Cpanel::CustomEventHandler;
```

This declaration instructs Perl to treat all of the file's functions as a part of the `Cpanel::CustomEventHandler` namespace.

For more information, read perldoc.perl.org's package documentation.

Set the strict pragma

```
use strict;
```

Standardized Hooks

About Standardized Hooks

- *The Standardized Hooks system triggers scripts or applications whenever a specific action is performed in cPanel & WHM.*
- *This functionality is useful for developers and system administrators.*

Compatible with:

- cPanel 11.32+
- WHM 11.32+

Related Documentation

- Manage Hooks
- Tweak Settings - Development — **Standardized Hooks - Debug Mode**

  The Standardized Hooks System's debug mode helps to troubleshoot hook issues. For more information, read our Guide to Standardized Hooks - Debug Mode documentation.

This declaration instructs Perl to return errors if the file contains potentially unsafe code.

For more information, read perldoc.perl.org's strict documentation.

Dependencies

```
use Cpanel::Logger ();
```

You may wish to include one or more dependencies in your `CustomEventHandler.pm` file.

- Use the `Cpanel::Logger` module to log error messages to a log file.
- Use the `Cpanel::API` module to call API functions.

> **Note:**
>
> The system performs custom event handler actions as the user who called the original API function. For this reason, you **must** ensure that all of the accounts on the server have permission to write to any files that receive custom event handler output.

For more information, read perldoc.perl.org's use documentation.

The `event()` subroutine

```
sub event {
    my ( $api_version, $type, $module, $func, $cfgref, $dataref
) = @_;
    if ($module eq 'ftp') {
        if ($func eq 'addftp') {
                # Do something for the addftp function.
        } elsif ($func eq 'delftp') {
        # Do something for the delftp function.
        }
    }
    return 1;
}
```

- Line 7 begins the `event()` subroutine.
- Line 8 declares the `event()` subroutine's variables.
- Lines 9 through 17 conditional statements to perform actions when the module or function match specific requirements.

> **Notes:**
>
> - Specific event function return values on `pre` events can block the associated function.
> - You can generate error messages in the `event()` subroutine.

The system passes the following parameters to the `event()` subroutine:

Parameter	Type	Description	Possible values	Example
`api_version`	*string*	The API function's cPanel API version.	• 1 — cPanel API 1 • 2 — cPanel API 2 • 3 — UAPI	2

You can select the following options:

- *Debug mode is off.* — The system does not display debug information or log it to the error log.
- *Debug mode is on. The system displays information about a hook while it executes, but does **not** log debug data to the error log.*
- *Debug mode is on. The system displays information about a hook while it executes **and** logs debug data to the error log.*

> **Imp**
>
> This
> **larg**
> amo

`type`	*string*	The step in the process at which the system runs the custom event handler. **Remember:** The system runs the `event()` subroutine before **and** after every API function.	• `pre` — The system will run the function next. • `post` — The system has already run the function.	`pre`
`module`	*string*	The API function's module.	A valid cPanel API 1, cPanel API 2, or UAPI module name.	`Email`
`event`	*string*	The API function's name.	A valid cPanel API 1, cPanel API 2, or UAPI function name.	`addforward`
`cfgref`	*hash reference*	A hash of the API call's input parameters and their values. **Note:** For UAPI functions, the system returns this hash reference in the $`args` variable.	Read the associated function's documentation for a list of input parameters.	`'fwdemail'`
`dataref`	*hash reference*	A hash of the API call's return data. The system only passes in this data for `post` custom event handlers. **Note:** For UAPI functions, the system returns this hash reference in the $`result` variable.	Read the associated function's documentation for a list of returns. If the API function does not return data, this parameter will contain an empty array.	`hashref`

Block functions

If the `event()` subroutine returns any of the following values for a `pre` event, the system will block (skip) the API call:

- `0`
- `undef`
- A negative number.
- A blank string.

Warning:

If the system blocks an API function due to a custom event handler's return, it will **not** produce an error unless you specifically generate one.

Error handling

Logged errors

To generate an error in the error log, print the desired error message to STDERR:

- *Debug mode is on. The system displays information about every stage for every hookable event, even if no hooks exist for that stage.*

This setting defaults to *Debug mode is off*.

Note:

```
print STDERR "This is an error message.";
```

You can also use the `Cpanel::Logger` module to direct error messages to specific log files.

The `new()` method

```
my $logger= Cpanel::Logger->new();
my $logger = Cpanel::Logger->new( { alternate_logfile => /path/to
/log_file } );
```

Use this method to instantiate the `Cpanel::Logger` object. By default, the `Cpanel::Logger` module logs error messages to the `error_log` file.

- Line 1 instantiates the `Cpanel::Logger` object with the default log file.
- Line 2 instantiates the `Cpanel::Logger` object, and instructs it to log error messages to the `/path/to/log_file` file. The `alternate_logfile` variable's value is the absolute path to the desired log file.

The `info()` method

```
$logger->info("This is an error message.");
```

Use this method to log an informational message and a time stamp.

For example, the line above produces the following log file entry:

```
YYYY-MM-DD HH:MM:SS info [cpanel] This is an error message.
```

The `warn()` method

```
$logger->warn("The server is on fire!");
```

Use this method to log an error message and a time stamp. The system will append a backtrace to the log entry, which will display the associated API calls.

For example, the line above produces the following log file entry:

```
[YYYY-MM-DD HH:MM:SS TZ] warn [cpanel] The server is on fire! at
/usr/local/cpanel/Cpanel/CustomEventHandler.pm line 31
    Cpanel::CustomEventHandler::event(2, 'pre', 'email',
'addpop', HASH(0x94df8b4), undef) called at /usr/local/cpanel
/Cpanel/EventHandler.pm line 63
    Cpanel::EventHandler::event(2, 'pre', 'email', 'addpop', HASH
(0x94df8b4)) called at cpanel.pl line 3970
    main::api2_exec('Email', 'addpop', HASH(0xb8f108c), HASH
(0x94df8b4)) called at cpanel.pl line 668
    main::docpanelaction(HASH(0xb8e2b80)) called at cpanel.pl
line 4915
    main::run_fast_json_mode() called at cpanel.pl line 341
```

Interface errors

> **Note:**
>
> This variable is only available for some API functions.

The `$Cpanel::CPERROR` variable can generate an error in the cPanel interface. This variable also requires the `$Cpanel::context` variable.

If, instead of this setting, you use the `/var/cpanel/debughooks` file to enable debug mode, your locale may revert to the English defaults for JavaScript elements. To fix this problem, run the following commands to disable debug mode and restart the `cpsrvd` daemon:

- Normally, the system sets the `$Cpanel::context` variable automatically.
- If the system does not automatically set the context, this variable contains the module's name in lowercase.

For example, the following line adds an error message:

```
$Cpanel::CPERROR{$Cpanel::context} = "This is an error message.";
```

Debug

To install a debugger version of the custom event handler, perform the following steps:

1. Download and install the `CustomEventHandler-Dumper.tar.gz` file.
2. Follow the instructions in the `README` file.

```
echo -n > /var/cpanel/debughooks/scripts/restartsrv_cpsrvd
```

- WHM API 1 Functions - reorder_hooks — This function changes the order of script hooks.
- WHM API 1 Functions - delete_hook — This function removes a script hook.
- WHM API 1 Functions - edit_hook — This function edits a script hook.