Tutorial - Use UAPI's Fileman::upload_files Function in Custom Code

Introduction

This tutorial explains the process to upload files with UAPI's `Fileman::upload_files` function. This function requires that you call it in a specific way to ensure that it uploads the file correctly.

Important:
We **strongly** recommend that you do **not** use cPanel API 1's `Fileman::uploadfiles` function **or** cPanel API 2's `Fileman::uploadfiles` function to upload files in custom code. These functions are deprecated.

Upload files in custom code

Perl PHP

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.

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

Set web request module dependencies.

We recommend that you set a dependency for one or more modules that allow you to perform web requests.

- This tutorial uses the `LWP::UserAgent` and `LWP::Protocol::https` modules.
- The `HTTP::Tiny` module also fulfills this need.

```
# Allow my code to perform web requests.
use LWP::UserAgent;
use LWP::Protocol::https;
```

In this tutorial

Related documentation

- Tutorial - Create a Standardized Hook
- Tutorial - Create a Custom cPanel Style

Use UTF-8 encoding.

You **must** use UTF-8 encoding when you call UAPI functions. If you do not use the correct encoding, your code may result in wide character warnings.

```
# Use the correct encoding to prevent
wide character warnings.
use Encode;
use utf8;
```

Use JSON formatting.

Declare the use of JSON formatting, which ensures that your code can properly decode JSON.

```
# Properly decode JSON.
use JSON;
```

Use Base64 encoding.

Declare the use of the `MIME::Base64` module, which ensures that your code properly interacts with Base64-encoded authentication headers.

```
# Function properly with Base64
authentication headers.
use MIME::Base64;
```

Provide authentication information.

Declare the variables for your cPanel account's username and password, which allow your code to authenticate with cPanel & WHM.

```
# Authentication information.
my $username = 'username';
my $password = '12345luggage';
```

Content by label

There is no content with the specified labels

Set the API call location.

Set the `$request` variable to the URL for the desired API call.

```
# The URL for the Fileman::upload_files
UAPI function.
my $request =
"https://localhost:2083/execute/Fileman
/upload_files";
```

Allow HTTPS connections to unsigned services.

Set the `PERL_LWP_SSL_VERIFY_HOSTNAME` environment variable to `0` to allow HTTPS connections to unsigned services.

Warning:
You **must** include this step in your code. Services on the local host are **alw ays** unsigned.

```
# Required to allow HTTPS connections to
unsigned services.
# Services on localhost are always
unsigned.
$ENV{PERL_LWP_SSL_VERIFY_HOSTNAME} = 0;
```

Create a new `UserAgent` object.

Use the `LWP::UserAgent` module's `new()` method to create a new `UserAgent` obj ect for the API call.

```
# Create a useragent object.
my $ua = LWP::UserAgent->new();
```

Create authentication headers.

Use the new `UserAgent` object to set up authentication headers that use the `$usern ame` and `$password` values from lines 22 and 23.

```
# Add authentication headers.
$ua->default_header(
    'Authorization' => 'Basic ' .
MIME::Base64::encode("$username:$passwo
rd"),
);
```

Add content to upload to the file.

Add the content that you wish to upload to the `$post_content` variable. In this example, that content is a simple string of HTML code.

```
# Add content to upload to the file.
my $post_content =
"<html><head>Header</head><body>Hello,
world!</body></html>";
```

Create a POST request for the call.

Create a POST request that includes the function's required input parameters. The `$response` variable will contain the call's output.

```
# Make the call.
my $response = $ua->post($request,
    Content_Type => 'form-data',
    Content => [
        dir => 'public_html',
        'file-1' => [
            undef,
            'nonindex.html',
            Content => $post_content,
        ],
    ],
);
```

Create an object to decode the call's output.

Create an object to decode the call's JSON-formatted output. You will use this object to sort and pretty-print the JSON.

```
# Create an object to decode the JSON.
# Sorted by keys and pretty-printed.
my $json_printer =
JSON->new->pretty->canonical(1);
```

UTF-8 encode and then decode the call's output.

Use the `Encode` module to UTF-8 encode the call's output in the `$response` variable . Then, use the `JSON` module to decode the output.

> **Note:**
> If you do not UTF-8 encode this output, you may receive wide character warnings.

```
# UTF-8 encode before decoding to avoid
wide character warnings.
my $content =
JSON::decode_json(Encode::encode_utf8($
response->decoded_content));
```

Print the function's output.

Use the object that line 58 created to pretty-print and sort the JSON output. Then, use the `Encode` module to UTF-8 encode the pretty-printed and sorted JSON, and print the final result.

> **Note:**
> If you do not UTF-8 encode this output, you may receive wide character warnings.

```
# Print output, UTF-8 encoded to avoid
wide character warnings.
print
Encode::encode_utf8($json_printer->enco
de($content));
```

Initiate error logging.

Set error reporting to `E_ALL` in order to log all errors and warnings.

Note:
You can omit this step in production code, but we **strongly** recommend its use during development.

```
// Log everything during development.
// If you run this on the CLI, set
'display_errors = On' in php.ini.
error_reporting(E_ALL);
```

Provide authentication information.

Declare the variables for your cPanel account username and password in order to allow your code to authenticate with cPanel & WHM.

```
// Declare your username and password
for authentication.
$username = 'example';
$password = 'luggage12345';
```

Define the API call.

Define the API host and the URL for the desired API call.

```
// Define the API call.
$cpanel_host = 'localhost';
$request_uri =
"https://$cpanel_host:2083/execute/File
man/upload_files";
```

Define the file to upload.

Define the filename that you wish to upload and its destination.

Note:
This example manually defines the filename and destination. You can also use functions or the `$argv` array to pass these values.

```
// Define the filename and destination.
$upload_file =
realpath("/path/to/fileto.upload");
$destination_dir = "public_html";
```

Set up the payload to send to the server.

Set up the payload to send to the server. This data includes the function's required input parameters.

Note:
This example uses code that is valid with PHP 5.5 and higher. For more information, read PHP.net's `curl_file_upload` documentation.

```
// Set up the payload to send to the
server.
if( function_exists( 'curl_file_create'
) ) {
    $cf = curl_file_create( $upload_file
);
} else {
    $cf = "@/".$upload_file;
}
$payload = array(
    'dir'    => $destination_dir,
    'file-1' => $cf
);
```

Set up the curl request object.

Set up a curl request object that uses the specified `$username` and `$password` vari ables.

Notes:

- Only use the `CURLOPT_SSL_VERIFYHOST` and `CURLOPT_SSL_VERIFYPEER` options with self-signed or expired certificates.
- `localhost` **always** uses a self-signed certificate.

```
// Set up the curl request object.
$ch = curl_init( $request_uri );
curl_setopt( $ch, CURLOPT_HTTPAUTH,
CURLAUTH_BASIC );
curl_setopt( $ch, CURLOPT_USERPWD,
$username . ':' . $password );
curl_setopt( $ch,
CURLOPT_SSL_VERIFYHOST, false );
curl_setopt( $ch,
CURLOPT_SSL_VERIFYPEER, false );
```

Create a POST request for the call.

Create a POST request that includes the function's required input parameters and uses the `$payload` variable from line 24.

```
// Set up a POST request with the
payload.
curl_setopt( $ch, CURLOPT_POST, true );
curl_setopt( $ch, CURLOPT_POSTFIELDS,
$payload );
curl_setopt( $ch,
CURLOPT_RETURNTRANSFER, true );
```

Make the call.

Use the `curl_exec()` method to call the API function. Then, use the `curl_close()` method to close the cURL object that you created.

```
// Make the call, and then terminate the
cURL caller object.
$curl_response = curl_exec( $ch );
curl_close( $ch );
```

Decode and validate the output.

Use the `json_decode()` method to decode the `$curl_response` value, which contains the call's output. Then, validate that output.

- Lines 47 through 49 of the example below check to ensure that the call returned output.

- Lines 50 through 52 of the example below ensure that there were no errors.

```
// Decode and validate output.
$response = json_decode( $curl_response
);
if( empty( $response ) ) {
    echo "The cURL call did not return
valid JSON:\n";
    die( $response );
} elseif ( !$response->status ) {
    echo "The cURL call returned valid
JSON, but reported errors:\n";
    die( $response->errors[0] . "\n" );
}
```

Print output.

Print the validated output and exit.

```
// Print and exit.
die( print_r( $response ) );
```

Completed code

When you finish this tutorial, your code will resemble the following examples:

Click to view the complete Perl code...

```
#!/usr/local/cpanel/3rdparty/bin/perl

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

# Allow my code to perform web
requests.
use LWP::UserAgent;
use LWP::Protocol::https;

# Use the correct encoding to prevent
wide character warnings.
use Encode;
use utf8;

# Properly decode JSON.
use JSON;
```

```
# Function properly with Base64
authentication headers.
use MIME::Base64;

# Authentication information.
my $username = 'username';
my $password = '12345luggage';

# The URL for the
Fileman::upload_files UAPI function.
my $request =
"https://localhost:2083/execute/Filem
an/upload_files";

# Required to allow HTTPS connections
to unsigned services.
# Services on localhost are always
unsigned.
$ENV{PERL_LWP_SSL_VERIFY_HOSTNAME} =
0;

# Create a useragent object.
my $ua = LWP::UserAgent->new();

# Add authentication headers.
$ua->default_header(
    'Authorization' => 'Basic ' .
MIME::Base64::encode("$username:$pass
word"),
);

# Add content to upload to the file.
my $post_content =
"<html><head>Header</head><body>Hello
, world!</body></html>";

# Make the call.
my $response = $ua->post($request,
    Content_Type => 'form-data',
    Content => [
        dir => 'public_html',
        'file-1' => [
            undef,
            'nonindex.html',
            Content => $post_content,
        ],
    ],
);

# Create an object to decode the
JSON.
# Sorted by keys and pretty-printed.
```

```
my $json_printer =
JSON->new->pretty->canonical(1);

# UTF-8 encode before decoding to
avoid wide character warnings.
my $content =
JSON::decode_json(Encode::encode_utf8
($response->decoded_content));

# Print output, UTF-8 encoded to
avoid wide character warnings.
```

```
print
Encode::encode_utf8($json_printer->en
code($content));
```

Click to view the complete PHP code...

```
<?php
// Log everything during development.
// If you run this on the CLI, set
'display_errors = On' in php.ini.
error_reporting(E_ALL);

// Declare your username and password
for authentication.
$username = 'example';
$password = 'luggage12345';

// Define the API call.
$cpanel_host = 'localhost';
$request_uri =
"https://$cpanel_host:2083/execute/Fi
leman/upload_files";

// Define the filename and
destination.
$upload_file =
realpath("/path/to/fileto.upload");
$destination_dir = "public_html";

// Set up the payload to send to the
server.
if( function_exists(
'curl_file_create' ) ) {
    $cf = curl_file_create(
$upload_file );
} else {
    $cf = "@/".$upload_file;
}
$payload = array(
    'dir'    => $destination_dir,
    'file-1' => $cf
);

// Set up the cURL request object.
$ch = curl_init( $request_uri );
curl_setopt( $ch, CURLOPT_HTTPAUTH,
CURLAUTH_BASIC );
curl_setopt( $ch, CURLOPT_USERPWD,
$username . ':' . $password );
curl_setopt( $ch,
CURLOPT_SSL_VERIFYHOST, false );
```

```
curl_setopt( $ch,
CURLOPT_SSL_VERIFYPEER, false );

// Set up a POST request with the
payload.
curl_setopt( $ch, CURLOPT_POST, true
);
curl_setopt( $ch, CURLOPT_POSTFIELDS,
$payload );
curl_setopt( $ch,
CURLOPT_RETURNTRANSFER, true );

// Make the call, and then terminate
the cURL caller object.
$curl_response = curl_exec( $ch );
curl_close( $ch );

// Decode and validate output.
$response = json_decode(
$curl_response );
if( empty( $response ) ) {
    echo "The cURL call did not
return valid JSON:\n";
    die( $response );
} elseif ( !$response->status ) {
    echo "The cURL call returned
valid JSON, but reported errors:\n";
    die( $response->errors[0] . "\n"
);
}
```

```
// Print and exit.
die( print_r( $response ) );
?>
```