Guide to cPanel Market Provider Modules

Introduction

Note:
We added this functionality in cPanel & WHM version 56.

Warnings:
We recommend that **only** advanced users use this feature.

This document explains how to create a third-party custom cPanel Market provider module with basic functionality. The cPanel Market system allows you to purchase and automatically install SSL certificates and other products through cPanel's *SSL/TLS Wizard* interface (*cPanel >> Home >> Security >> SSL/TLS Wizard*). cPanel & WHM servers ship with the cPanel Store provider module.

Third-party vendors can create their own provider modules in Perl in the `/var/cpanel/perl/Cpanel/Market/Provider/` directory and should use the `Cpanel::Market::Provider` namespace. For example, the Example certificate provider would create the `/var/cpanel/perl/Cpanel/Market/Provider/Example.pm` module in the `Cpanel::Market::Provider::Example` namespace.

Module function interfaces

Third-party developers **must** implement the following functions in order to create a functional cPanel Market provider module:

- convert_identity_verif
- convert_ssl_identity_v
- create_shopping_cart
- get_checkout_url
- get_login_url
- get_products_list
- get_ssl_certificate_if_a
- get_support_uri_for_o
- prepare_system_for_d
- request_ssl_certificate
- set_url_after_checkou
- undo_domain_control_
- validate_login_token
- validate_request_for_c

`convert_identity_verification_to_csr_subject`

This function retrieves the `identity_verification` information from the item description and returns the output that the `csr` subject will contain.

Parameters

A list of `key=value` pairs:

- Returns a list of `key=value` array references, which the system adds to the `csr` subject.

Returns

The information provided in the `identity_verification` hash from the `request_ssl_certificates` function.

Equivalent UAPI Function

None.

`convert_ssl_identity_verific`

Market Provider Modules

Related Documentation

- Market Provider Manager
- Market Provider Manager

`ation_to_order_item_parameters`

This function retrieves the `identity_verification` information from the item description and returns the output that the `item` parameters will contain.

Parameters

A list of `key=value` pairs:

- Returns a list of `key=value` array references; these are added to the CSR's `item` parameters.

Returns

The information provided in the `identity_verification` hash from the `request_ssl_certificates` function.

Equivalent UAPI Function

None.

`create_shopping_cart`

This function sends an order to the provider.

Parameters

A list of `key=value` pairs:

- **Required:** An `access_token` from the `validate_login_token` function's results.
- `items` — A reference to an array of hash references, each of which contains the following values:
  - `product_id` — The ID of the item to order.
  - Any other `key=value` pairs that a specific product ID may require.

Returns

A list of the following values:

- The `order_id` value.
- A reference to an array of hash references, each of which **must** include o

rder_item_id values in the same order that they appeared in the input items array.

Equivalent UAPI Function

`Market::create_shopping_cart`

get_checkout_url

This function retrieves the checkout URL for the Market provider.

Parameters

The order ID, which the `create_shopping_cart` function returns.

Returns

The URL that the browser should access to process payment.

Equivalent UAPI Function

None.

get_login_url

This function retrieves the login URL for the Market provider.

Parameters

The URL to which to redirect to after login.

- This URL may contain a query parameter.
- The login redirect **must** append a code parameter to the query string. The `validate_login_token` function requires this code value as its login token.

> **Note:**
> The query string uses www-form-URIencoded format (HTML 4.01/17.13.1).

- WHM API 1 Functions - validate_login_token — This function validates a login token with the cPanel Store or a cPanel Market provider, and then returns access tokens.

Returns

The URL that the browser should access to allow the user to log in.

Equivalent UAPI Function

`Market::get_login_url`

`get_products_list`

This function lists all of the available Market provider's products.

Parameters

None.

Returns

A list of hashes with information about each product.

- Each item **must** include a `product_id`.

Equivalent UAPI Function

`Market::get_all_products`

`get_ssl_certificate_if_available`

This function gets the certificate from the provider and returns a status code.

Parameters

The item's `order_item_id`.

Returns

A hash of the following values:

- `certificate_pem` — The certificate in PEM format, or `undef` if the certificate is not yet available.
- `status_code` — The status of the certificate, which contains one of the following values:
  - `CertificateNotFound`
  - `RequiresApproval`

- WHM API 1 Sections - Market — Market functions allow you to manage cPanel Market providers and products available to your users.

- `OrderCanceled`
- `OrderItemCanceled`

Equivalent UAPI Function

`Market::get_ssl_certificate_if_available`

`get_support_uri_for_order`

This function retrieves the Support URI for the Market Provider if the customer needs to cancel an order or request assistance.

Parameters

A list of `key=value` pairs:

- `order_id` — The order ID, which the `create_shopping_cart`function returns.
- `order_item_id` - The ID of the individual item in the order, which the `create_shopping_cart` function returns.

Returns

A string that contains the URI to contact for support.

Equivalent UAPI Function

None.

`prepare_system_for_domain_control_validation`

This function performs all necessary steps to prepare the system for Domain Control Validation (DCV) for a SSL certificate.

Parameters

A list of `key=value` pairs:

- `product_id` — The SSL certificate item's ID.
- `csr` — The certificate signing request (CSR) in PEM format.

Returns

None.

Equivalent UAPI Function

None.

`request_ssl_certificates`

This function submits a request for a certificate order to the provider.

Parameters

A list of `key=value` pairs:

- `access_token` — The access token for the session.
- `url_after_checkout` — The post-checkout URL.
- `identity_verification` — A hash that contains the required information for an EV or OV certificate.
- A JSON-encoded string that contains certificate details.

Returns

A list of the following values:

- The `order_id` value.
- The `checkout_url` value.
- A hash which **must** include `order_item_id` and `key_id` values.

Equivalent UAPI Function

`Market::request_ssl_certificates`

`set_url_after_checkout`

This function sets the URL to which the checkout logic will redirect the user after they check out.

Parameters

A list of `key=value` pairs:

- `order_id` — The order ID, which the `create_shopping_cart` function returns.
- `access_token` — The access token, which the `validate_login_token` function returns.

- `url_after_checkout` — The post-checkout URL.

Returns

None.

Equivalent UAPI Function

`Market::set_url_after_checkout`

`undo_domain_control_validation_preparation`

This function reverses changes that the `prepare_system_for_domain_control_validation` function makes.

Parameters

A list of `key=value` pairs:

- `product_id` — The SSL certificate item's ID.
- `csr` — The CSR in PEM format

Returns

None.

Equivalent UAPI Function

None.

`validate_login_token`

This function validates a login token to a Market provider and returns an access token.

Parameters

- The login token, which the `get_login_url` function returns in the "code" query string variable.
- The value previously passed to the `get_login_url` function.

Returns

A hash with the `access_token` result, which is

the API access token.

Equivalent UAPI Function

`Market::validate_login_token`

`validate_request_for_one_item`

This function returns an exception if an error occurs.

Parameters

A list of `key=value` pairs, which the `create_shopping_cart` function received:

- `product_id` — The product ID, as referenced in the product list.
- Any other `key=value` pairs that a specific product ID may require.

Returns

None.

Equivalent UAPI Function

None.

Constants

We recommend that third-party developers declare the following constants:

Constant	Description	Example
`REQUEST_URI_DCV_PATH`	The path to DCV check file, relative to the document root directory.	`'^/[A-F0-9]{32}\\.txt(?: Sectigo DCV)?$',`
`URI_DCV_ALLOWED_CHARACTERS`	The characters that the provider allows in the filename that it uses to check for DCV.	`[ 0 .. 9, 'A' .. 'Z' ],`
`URI_DCV_RANDOM_CHARACTER_COUNT`	The number of characters that the provider allows in the DCV check filename.	`32,`
`EXTENSION`	The DCV check file's extension that the provider requires.	`'txt'`

`DCV_USER_AGENT`	The user agent string that the system uses for the imitated local DCV check.	`'SECTIGO DCV'`

cPanel Market purchase workflow

Third-party provider modules **must** use the following workflow:

1. The user prepares a shopping cart in the browser with information that the provider's custom Perl module returns through the `get_products_list` function.
2. When the user wants to check out, the cPanel application redirects the user to the login page that the UAPI Function `get_login_url` function provides.
3. After the user logs in, the login server redirects the user back to the cPanel application with a code in the URL query parameter. The cPanel application calls the `validate_login_token` function in order to send that code to the provider and obtain an access token.
4. The cPanel application prepares a shopping cart with the access token:
    a. The system validates the requests with the provider with the `create_shopping_cart` function. You can insert custom logic with the `validate_request_for_one_item` function.
    b. The `create_shopping_cart` function returns an `order_id` value that the cPanel application will use to maintain state after payment.
    c. The `create_shopping_cart` function also returns a unique `order_item_id` value for each item that the customer orders.
    d. If the user purchased a certificate, their server begins to poll the provider for that certificate's `order_item_id` value.
5. The cPanel application calls the `set_url_after_checkout` function to set a post-checkout redirection URL.
6. The user's server redirects the user to a checkout URL that the `get_checkout_url` function provides.
7. The provider processes payment, but if the order is for a certificate, they do **not** finalize the charge.
8. The provider redirects the user back to the cPanel application (for example, the URL that the `set_url_after_checkout` function previously set).

Certificate orders

For certificate orders, the module performs the following additional steps:

1. The cPanel application calls the `get_ssl_certificate_if_available` function at regular intervals to poll the provider for the certificate.
2. After the provider issues the certificate, the provider finalizes payment.

> **Note:**
> We recommend that you finalize payment **after** you issue the certificate in order to avoid unnecessary additional chargeback fees. If the user deletes the Domain Control Validation (DCV) file from their account before the DCV process happens, this effectively cancels the order.

3. If a poll is successful, the cPanel application downloads and installs the SSL certificate, and then it stops polling.

Example

The following custom provider module outline demonstrates a minimal set of functionality.

> **Warning:**
> This example does **not** reflect a fully-functional module and only

demonstrates a basic workflow. Your implementation requires more internal logic. Also, this example does **not** demonstrate the necessary API functions that would allow your module to hook into your store.

Click to view...

```
package
Cpanel::Market::Provider::Example;

use strict;
use warnings;
use autodie;

use constant {
    #The last bit here doesn't
actually happen; we just want to
    #tag the regexp so that it's very
clear where it came from since
    #otherwise this pattern is
somewhat generic.
    REQUEST_URI_DCV_PATH           =>
'^/[A-F0-9]{32}\\.txt(?: Sectigo
DCV)?$',
    URI_DCV_ALLOWED_CHARACTERS     =>
[ 0 .. 9, 'A' .. 'Z' ],
    URI_DCV_RANDOM_CHARACTER_COUNT =>
32,
    EXTENSION                      =>
'txt',
};

use HTTP::Tiny ();

sub _DISPLAY_NAME { return 'Example's
stuff' }

#---------------------------------
---------------
### STEP 1 - The products list.
### example1 is a certificate, which
will exist in the ssl_certificate
product group.
### example2 is something other than
a certificate, which will exist in
the example_things product group.
sub get_products_list {
    return(
        {
            product_id => 'example1',
            description => 'A
certificate',
            display_name => 'Example
Cert',
            price_unit => 'USD',
```

```
            product_group =>
'ssl_certificate',
            x_price_per_domain =>
100,
            x_ssl_per_domain_pricing
=> 1
        },
        {
            product_id => 'example2',
            description => 'A thing',
            display_name => 'Thing
#2',
            price => 5,
            price_minimum => 4,
            price_unit => 'USD',
            product_group =>
'example_things',

x_price_per_wildcard_domain => 200,

x_price_per_wildcard_maximum => 1000,

x_price_per_wildcard_minimum => 100
        },
    );
}

#-----------------------------------
---------------
### STEP 2 - Get the login URL.

sub get_login_url {
    my ($after) = @_;

    return
'http://example.com/cpmarket_login.ht
ml?' .
HTTP::Tiny->www_form_urlencode(
        {
           ### This is where we input
the redirection URI
           redirect_uri => $after,
        },
    );
}

#-----------------------------------
---------------
### STEP 3 - Let's validate the code
and send the code to the provider get
the token

sub validate_login_token {
```

```
    my ($token) = @_;

    return { access_token =>
"access_$token" };
}

#-----------------------------------
---------------
### STEP 4a - If you want to validate
the request for an item.
### Validation of the item depends on
your product list and store API
structure.
### This subroutine intentionally
left blank.

sub validate_request_for_one_item {}

### STEP 4a - Create the shopping
cart.

sub create_shopping_cart {
    my (%opts) = @_;

 #Usually this will come from a
remote service that manages the
shopping cart.
 #For the purposes of this
demonstration module, however, we
generate a random order_id.
 #The order ID can be any
provider-unique string or numeric
value;
 #i.e., no two orders may ever have
the same order ID.
 my $order_id = int rand 100000;

 return(
        ### STEP 4b - Your provider
generates the true order_id. We use
"int rand 100000" to skip that step.
        int( rand 100000 ),
        [
            {
                ### Each item in the
cart must receive a provider-unique
order_item_id.
    ### A given provider may only
ever use a given order_item_id once.
    order_item_id => sprintf('%04x',
int rand 65536),
            },
        ],
```

```
    );
}

#-----------------------------------
---------------
### STEP 5 - Tell the checkout where
we should go after checking out.

sub set_url_after_checkout {
    my (%opts) = @_;

    #This needs custom code that will
use the "access_token"
    #to set the "url_after_checkout
for the given "order_id".
}

#-----------------------------------
---------------
### STEP 6 - Redirect to the checkout
URL.

sub get_checkout_url {
    my ($after) = @_;

    return
'http://example.com/cpmarket_checkout
.html?' .
HTTP::Tiny->www_form_urlencode(
        {
            order_id => $after,
        },
    );
}

#-----------------------------------
---------------

sub get_support_uri_for_order_item {
    my (%opts) = @_;
    my $query =
HTTP::Tiny->www_form_urlencode(
        { map { ( $_ => $opts{$_} ) }
qw( order_id  order_item_id ) },
    );
    return
"http://help.me.rhombus?$query";
}

#-----------------------------------
---------------
### LOGIC SPECIFIC TO SELLING SSL
CERTIFICATES
```

```
#-----------------------------------
---------------

sub get_ssl_certificate_if_available
{
    my ($order_item_id) = @_;

    #Check for the certificate; if
it's available, return it (PEM
format).
 #Otherwise...

 return undef;
}

### Domain control validation depends
on how your Certificate Authority
### confirms ownership of the domain.
For example, Sectigo’s validation
### looks for specific content at a
specific, publicly-accessible URL;
### both the content and the URL are
functions of the CSR.

### If you are not doing automatic
DCV, (e.g., you're requiring the end
### user to respond to an email) then
you can safely leave these functions
blank.

sub
prepare_system_for_domain_control_val
idation {
 my (%opts) = @_; #product_id, csr
}

sub
undo_domain_control_validation_prepar
ation {
 my (%opts) = @_; #product_id, csr
}

### The following functions are
available to retrieve information
from the
### item description. The retrieved
information is then added to either
the
### CSR subject or the item
parameters.

sub
convert_ssl_identity_verification_to_
```

```
csr_subject {
    my ($product_id, %id_ver) = @_;
    my @keys_in_csr = grep {
_is_in_csr($_) } keys %id_ver;
    #Any kind of manipulation of the
information could be done here.
    return map { [ $_ => $id_ver{$_}
] } @keys_in_csr;
}

sub
convert_ssl_identity_verification_to_
order_item_parameters
    my ($product_id, %id_ver) = @_;
    my @keys_not_in_csr = grep {
!_is_in_csr($_) } keys %id_ver;
    #Any kind of manipulation of the
information could be done here.
    return map { $_ => $id_ver{$_} }
```

```
  @keys_not_in_csr;
  }
```