Carrier modules - functions, creation and configuration

Carrier modules: functions, creation and configuration

This article was written by Fabien Serny, and published on September 28th, 2011 on the PrestaShop blog.

Foreword

Please note, this article is aimed at advanced developers (knowledge of hooks, the Carrier object...) and applies to PrestaShop 1.4 and later.
To find out what a hook is, I recommend reading the article by Julien Breux here: Understanding and using hooks

On a basic level PrestaShop lets you create carriers and configure how shipping fees are calculated based on weight and price ranges that you can specify in the back office.

The limits of this system are reached when you want to retrieve prices via webservices (UPS, USPS, Fedex) or want to make your own system for calculating shipping fees (by number of items in the basket, restricting yourself to one carrier or one or more countries, etc.).

This is when carrier modules come into play.

Where do I start?

There is a basic module you can download here. The article will start from this basic module but of course you can modify it and adapt it to your needs

In our example, you can manage two carriers and set an additional cost for them in addition to the price by range configured in the back office.

Certain parts of the code are deliberately duplicated to help you understand the mechanism. It is up to you to manage your specific list of carriers (using a data table for example).

Explanation of the example

First, please note the fact that the module belongs to the CarrierModule class and not the Module class. For now, let’s make a short list of the methods contained in this module and the way it works in general.

1) The __construct, install and uninstall methods

Construct:

```
// The carrier id is automatically filled in with the right id for the carrier.
// We will look at the purpose of this a little later.
public  $id_carrier;

private $_html = '';
private $_postErrors = array();
private $_moduleName = 'mycarrier';
/*
** Construct Method
**
*/
public function __construct()
{
       // Variables common to all modules (no need to present them)
    $this->name = 'mycarrier';
    $this->tab = 'shipping_logistics';
    $this->version = '1.0';
    $this->author = 'YourName';
    $this->limited_countries = array('fr', 'us');
    parent::__construct ();
    $this->displayName = $this->l('My Carrier');
    $this->description = $this->l('Delivery methods that you want');
        // If the module is installed, we run a few checks
    if (self::isInstalled($this->name))
    {
        // We retrieve the list of carrier ids
        global $cookie;
        $carriers = Carrier::getCarriers($cookie->id_lang, true, false, false,
                NULL, PS_CARRIERS_AND_CARRIER_MODULES_NEED_RANGE);
        $id_carrier_list = array();
        foreach($carriers as $carrier)
            $id_carrier_list[] .= $carrier['id_carrier'];



        // We look to see if the carriers have been created for the module
        // And if any additional fees have been configured
        // These warnings will appear on the page where the modules are listed
        $warning = array();
        if (!in_array((int)(Configuration::get('MYCARRIER1_CARRIER_ID')),
                  $id_carrier_list)) $warning[] .= $this->l('"Carrier 1"').' ';
        if (!in_array((int)(Configuration::get('MYCARRIER2_CARRIER_ID')),
               $id_carrier_list)) $warning[] .= $this->l('"Carrier 2 Overcost"').' ';
        if (!Configuration::get('MYCARRIER1_OVERCOST'))
            $warning[] .= $this->l('"Carrier 1 Overcost"').' ';
        if (!Configuration::get('MYCARRIER2_OVERCOST'))
            $warning[] .= $this->l('"Carrier 2 Overcost"').' ';
        if (count($warning))
    $this->warning .= implode(' , ',$warning).$this->l('must be <br />configured');
    }
}
```

Install:

```
/*
** Install / Uninstall Methods
**
*/
public function install()
{
    // We create a table containing information on the two carriers
   // that we want to create

    $carrierConfig = array(
    0 => array('name' => 'Carrier1',
      'id_tax_rules_group' => 0, // We do not apply thecarriers tax
      'active' => true, 'deleted' => 0,
      'shipping_handling' => false,
      'range_behavior' => 0,
      'delay' => array(
        'fr' => 'Description 1',
        'en' => 'Description 1',
        Language::getIsoById(Configuration::get
                ('PS_LANG_DEFAULT')) => 'Description 1'),
      'id_zone' => 1, // Area where the carrier operates
      'is_module' => true, // We specify that it is a module
      'shipping_external' => true,
      'external_module_name' => 'mycarrier', // We specify the name of the module
      'need_range' => true // We specify that we want the calculations for the ranges
    // that are configured in the back office
      ),
     1 => array('name' => 'Carrier2',
      'id_tax_rules_group' => 0,
      'active' => true,
      'deleted' => 0,
      'shipping_handling' => false,
      'range_behavior' => 0,
      'delay' => array(
        'fr' => 'Description 1',
        'en' => 'Description 1',
        Language::getIsoById(Configuration::get<br />
               ('PS_LANG_DEFAULT')) => 'Description 1'),
      'id_zone' => 1,
      'is_module' => true,
      'shipping_external' => true,
      'external_module_name' => 'mycarrier',
      'need_range' => true
    ),
);
    // We create the two carriers and retrieve the carrier ids
    // And save the ids in a database
    // Feel free to take a look at the code to see how installExternalCarrier works
    // However you should not normally need to modify this function

    $id_carrier1 = $this->installExternalCarrier($carrierConfig[0]);
    $id_carrier2 = $this->installExternalCarrier($carrierConfig[1]);
    Configuration::updateValue('MYCARRIER1_CARRIER_ID', (int)$id_carrier1);
    Configuration::updateValue('MYCARRIER2_CARRIER_ID', (int)$id_carrier2);
    // Then proceed with a standard module install
    // Later we will take a look at the purpose of theupdatecarrier hook

    if (!parent::install() ||
        !Configuration::updateValue('MYCARRIER1_OVERCOST', '') ||
        !Configuration::updateValue('MYCARRIER2_OVERCOST', '') ||
        !$this->registerHook('updateCarrier'))
        return false;

    return true;
}
```

Uninstall:

```
public function uninstall()
{
    // We first carry out a classic uninstall of a module
    if (!parent::uninstall() ||
        !Configuration::deleteByName('MYCARRIER1_OVERCOST') ||
        !Configuration::deleteByName('MYCARRIER2_OVERCOST') ||
        !$this->unregisterHook('updateCarrier'))
        return false;
    // We delete the carriers we created earlier
    $Carrier1 = new Carrier((int)(Configuration::get('MYCARRIER1_CARRIER_ID')));
    $Carrier2 = new Carrier((int)(Configuration::get('MYCARRIER2_CARRIER_ID')));
// If one of the two modules was the default carrier,
//we choose another
    if (Configuration::get('PS_CARRIER_DEFAULT') == (int)($Carrier1->id) ||
        Configuration::get('PS_CARRIER_DEFAULT') == (int)($Carrier2->id))
    {
        global $cookie;
        $carriersD = Carrier::getCarriers($cookie->id_lang, true, false, false, <br />
                NULL, PS_CARRIERS_AND_CARRIER_MODULES_NEED_RANGE);
        foreach($carriersD as $carrierD)
            if ($carrierD['active'] AND !$carrierD['deleted']
                AND ($carrierD['name'] != $this->_config['name']))
                Configuration::updateValue('PS_CARRIER_DEFAULT',<br />
                                $carrierD['id_carrier']);
    }
    // Then we delete the carriers using variable delete
    // in order to keep the carrier history for orders placed with them

    $Carrier1->deleted = 1;
    $Carrier2->deleted = 1;
    if (!$Carrier1->update() || !$Carrier2->update())
        return false;

    return true;
}
```

2) Back office methods

I will not dwell on most of the back office methods (getContent, _displayForm, _postValidation and _postProcess), which are relatively simple and are there to enable you to set up carrier costs. I will let you discover them in the example module.

However, I would like to draw your attention to the hookupdateCarrier method below. In PrestaShop, every time you edit a carrier, the carrier is automatically archived and a new carrier is created.

Technically, PrestaShop changes the deleted flag of the carrier to 1 and create a new one.
This is why when you edit a carrier via the "Carriers" tab in your back office, its id changes.

You therefore have to hook the module in order to update the carrier id when it changes.

```
/*
** Hook update carrier
**
*/
public function hookupdateCarrier($params)
{
    // Update the id for carrier 1
    if ((int)($params['id_carrier']) == (int)(Configuration::get<br />('MYCARRIER1_CARRIER_ID')))
        Configuration::updateValue('MYCARRIER1_CARRIER_ID', (int)<br />
                ($params['carrier']->id));
    // Update the id for carrier 2
    if ((int)($params['id_carrier']) == (int)(Configuration::get<br />('MYCARRIER2_CARRIER_ID')))
        Configuration::updateValue('MYCARRIER2_CARRIER_ID', (int)<br />
               ($params['carrier']->id));
}
```

3) Front office methods

```
/*
** Front Methods
**
** If you set the variable need_range to true when you created your carrier
** in the install() method, the method called up by the Cart class
** will be getOrderShippingCost()
** Otherwise the method called up will be getOrderShippingCostExternal
**
** The $params variable contains the basket, the customer and their addresses
** The $shipping_cost variable contains the cost calculated
** according to the price ranges set
** for the carrier in the backoffice
**
*/

public function getOrderShippingCost($params, $shipping_cost)
{
    // This example returns the shipping fee with the additional cost
    // but you can call up a webservice or perform the calculation you want
    // before returning the final shipping fee

    if ($this->id_carrier == (int)(Configuration::get('MYCARRIER1_CARRIER_ID')) &&
        Configuration::get('MYCARRIER1_OVERCOST') > 1)
        return (float)(Configuration::get('MYCARRIER1_OVERCOST'));

    if ($this->id_carrier == (int)(Configuration::get('MYCARRIER2_CARRIER_ID')) &&
        Configuration::get('MYCARRIER2_OVERCOST') > 1)
        return (float)(Configuration::get('MYCARRIER2_OVERCOST'));
    // If the carrier is not recognised, just return false
    // and the carrier will not appear in the carrier list

    return false;
}
public function getOrderShippingCostExternal($params)
{
    // This example returns the additional cost
    // but you can call up a webservice or perform the calculation you want
    // before returning the final shipping fee

    if ($this->id_carrier == (int)(Configuration::get('MYCARRIER1_CARRIER_ID')) &&
        Configuration::get('MYCARRIER1_OVERCOST') > 1)
                return (float)(Configuration::get('MYCARRIER1_OVERCOST'));
    if ($this->id_carrier == (int)(Configuration::get('MYCARRIER2_CARRIER_ID')) &&
            Configuration::get('MYCARRIER2_OVERCOST') > 1)
                return (float)(Configuration::get('MYCARRIER2_OVERCOST'));
    / If the carrier is not recognised, just return false
    // and the carrier will not appear in the carrier list

    return false;
}
```

Taking it further

The example here is very simple, but you can create more complex modules by associating them with webservices for example (as is the case with the UPS modules), or perform a calculation based on the number of products in your cart.

In short, you've now got the hang of calculating shipping fees 😅