

H-Sphere Developer Guide

Welcome to H-Sphere developer's guide. It explains how to add custom resources using H-Sphere API.

- [Adding Custom Merchant Gateways](#)
- [Running H-Sphere in Debug Mode](#) (Added: 21 Mar 2003)
- [Getting User Information from the H-Sphere Database](#) (Added: 26 Mar 2003)
- [Creating Plan Wizards with XML](#) (Added: 27 Mar 2003)

Adding Merchant Gateways

- [Creating merchant java classes](#)
- [Adding merchants to the system](#)

Creating Merchant Java Classes

To add a custom merchant gateway, you need to subclass the public abstract class **GenericMerchantGateway**. Your new class must implement its abstract methods:

- *init* – initializes the connection with a merchant gateway.
- *authorize* – verifies that the credit card is valid and has enough balance for the transaction;
- *capture* – confirms that the purchase has been completed and ready to be settled. This method can be called only after the *authorize* method;
- *charge* – charges an amount from the credit card without prior authorization;
- *voidAuthorize* – cancels the transaction that has not yet settled. This method can be called only after the *authorize* method.

Depending on the type of the resource, H-Sphere uses two schemes for charging users:

1. Immediate – used when no payments to third parties are required. In this scheme, only *charge* method is involved.
2. Two-step – used when the transaction involves payments to third parties, such as OpenSRS. This scheme involves the *authorize*

method on the first step and *capture* or *voidAuthorize* on the second. The transaction is performed in between the first and the second steps.

Depending on the merchant gateway, you will have to implement one of the following payment interfaces:

If the online processing center	Your class needs to	Example:
offers native Java libraries	call methods of the merchant's Java module	VeriSign
uses https protocol	send https requests and process https replies	Authorize.net
does not use Java (e.g. perl, C++, etc.)	call your own wrapper that will convert Java calls into its native language and back.	CyberCash

To download sample Java libraries, click [here](#).

Note: Use account ID (not user ID) as a parameter to pass to the remote server.

Now you can add this merchant gateway to the system.

Adding Merchants to the System

Create a template for adding and editing your merchant gateway properties. Follow the rules laid out in the [Advanced Customization](#) page of this manual.

You would have to create three files following these examples:

```
common/admin/merchant/AuthorizeNet.html
```

```
common/control/admin/merchant/AuthorizeNet.html
```

```
common/submit/admin/merchant/AuthorizeNet.sbm
```

and put them in the same directories as the examples.

1. Adding submit template

The submit template (.sbm) is responsible for passing parameters to the remote server and has two functions:

`mm.update` updates values for the current gateway.

syntax: `mm.update(gateway_id, VAR_NAME1, VAR_VALUE1, VAR_NAME2, VAR_VALUE2,)`

`gateway_id` – you should be able to get it from request, from previous form.

name/value pairs are used to configure the gateway.

`mnew`) create new gateway.

syntax: `mnew(, VAR_NAME1, VAR_VALUE1, VAR_NAME2, VAR_VALUE2,)`

This is an example of a control template:

```
<tr bgcolor="{LIGHT_STRIP}">
<td>
<call draw_label(lang.label.title)></td>
<td><input type="text" name="value1" value="{if
merchant}{merchant.VALUE1}<else>default value</if}">
</td>
</tr>
```

This line must be provided for every parameter.

3. Adding a template frame

A prototype of a template frame is `common/admin/merchant/AuthorizeNet.html`

Template frames are holders for template controls and are typically the same for all merchant gateways, so it will suffice to follow the prototype example.

4. Editing view.html

The last thing to do is to add your merchant gateway as a value to the drop-down list of available merchant gateways.

Create a custom template to substitute `common/admin/merchant/view.html`. See the [Advanced Customization](#) guide for instructions.

In the custom template, locate the select with name `template_name` and add the option for your merchant:

```
<option value="admin/merchant/MyCustomGateway.html">{lang....
```

Now you must be able to add this merchant gateway using the control panel web interface.

Running H-Sphere in Debug Mode

(versions 2.3 and higher)

To run H-Sphere in debug mode, do the following:

1. Compile classes with debug information:

```
./configure -javac --with-params="-g"  
make shiva
```

2. Set the following option "on" in jserv.conf to be able to launch JServ manually:

```
ApJServManual on
```

3. Create debug.sh shell script to run JServ in debug mode:

```
#!/bin/sh  
properties=/home/shiva/apache/etc/jserv.properties  
log=/home/shiva/apache/logs/jserv_manual.log  
CLASSPATH=$CLASSPATH:/usr/local/java/JSDK2.0/lib/jsdk.jar  
CLASSPATH=$CLASSPATH:/usr/local/apache/libexec/ApacheJServ.jar  
java -Xdebug -Xrunjdwpt:transport=dt_socket,address=9999,server=y,suspend=n \  
org.apache.jserv.JServ $properties $1 2 >> $log  
# address=9999 - port to which the debugger may be attached  
# choose any available port you like
```

4. In IDE, set the following debug configuration:

```
Transport: Socket  
Debugger Mode: Attach  
Host: localhost  
Port: 9999
```

5. Start Apache and run debug.sh after Apache start. Set IDE breakpoints and launch debug from IDE.

Getting User Information from the H-Sphere System Database

Here you will learn how to obtain different user data from system database. Make sure to run PgSQL database first, before running queries.

Getting user-specific information from the system database

- [User Info](#)
- [Mail Accounts Info](#)
- [Resellers Info](#)
- [Credit Card Info](#)
- [Virtual FTP Info](#)
- [User Home Directory](#)
- [User Website Directory](#)
- [User Hosting Plan](#)
- [User Domain Name](#)
- [User IP Address](#)
- [User IP Address by the Domain](#)
- [User Account ID by the Domain](#)
- [Logical Server by the Domain](#)
- [Shared IP by the Domain](#)
- [User E-mail Address](#)
- [User Billing Info](#)
- [Logical and Physical Servers](#)

Getting user-specific information from the system database

Selecting user info:

```
select users.* from users, user_account where users.id=user_account.user_id and  
user_account.account_id= n  
(n = user account id)
```

Selecting mail accounts info:

```
select m.* from mailboxes m, parent_child p where p.child_type=1002 and p.child_id=m.id and p.account_id= n
```

(n = user account id)

Selecting resellers info:

```
select u.* from users u, resellers r, user_account a where u.id=r.admin_id and r.id=a.user_id and account_id= n
```

(n = user account id)

Selecting credit card info:

```
select c.* from users u, user_billing_infos u_b, credit_card c, user_account a where u.id = u_b.user_id and c.id = u_b.billing_info_id and u.id=a.user_id and a.account_id= n
```

(n = user account id)

Selecting virtual FTP info:

```
select f.* from ftp_vuser f, parent_child p where p.child_type=2003 and p.child_id=f.id and p.account_id= n
```

(n = user account id)

Selecting user's home directory

To select user's home directory from the system database and connect it to account:

```
select * from unix_user where unix_user.id = parent_child.child_id and parent_child.account_id = n
```

(n = user account id)

Selecting user's website directory

Website directory always equals user's home directory and domain name.

Selecting user's hosting plan

To select the name of the plan, run the following query:

```
select plans.description from plans, accounts where accounts.id = n and plans.id =
accounts.plan_id;
(n = user account id)
```

Selecting domain name

To obtain data on domain names, run the following query:

```
select domains.name from domains, parent_child where domains.id = parent_child.child_id and
parent_child.account_id = n
(n = your id)
```

Selecting IP address

To retrieve data on all IPs, run the following query:

```
select IP from l_server_ips, parent_child where child_id = l_server_ips.r_id and
parent_child.child_type = 8 and parent_child.account_id = n
(n = your id).
```

There can be multiple IPs per domain.

Selecting IP of the domain:

To retrieve IP for the given domain name, run the following command:

```
select IP from l_server_ips, domains, parent_child where domains.id = parent_child.parent_id
and parent_child.child_id = l_server_ips.r_id and domains.name = 'YOUR_DOMAIN_NAME'
(enter your domain name)
```

Selecting account ID for the domain:

To get the data about account ID of the given domain, execute the query:

```
select account_id from domains, parent_child where domains.id = parent_child.child_id and
domains.name = 'DOMAIN_NAME'
```

Selecting logical server of the domain:

To retrieve IP for the given domain name, run the following command:

```
select hostid from unix_user, parent_child where parent_child.account_id = YOUR_ACCOUNT_ID
and child_id = unix_user.id
```

Selecting shared IP of the domain:

To get the information about shared IPs for the domain, run:

```
select IP from l_server_ips where where l_server_id ='HOSTID' and flag =' SHARED_IP'
```

selecting user's e-mail address

To receive data on clients contact emails, run the following query:

```
select * from contact_info, accounts where contact_info.id = accounts.ci_id and accounts.id
= n
(n = user account id);
```

Selecting user's billing info

To select billing info from accounts, run the following query:

```
select * from billing_info, accounts where billing_info.id = accounts.bi_id and accounts.id
=n
(n = user account id);
```

Verify users

To verify whether a user exists in H-Sphere:

```
select id from users where username='USER_NAME' and password='PASSWORD'
(enter user name and password)
```

Defining physical/logical servers

Run the following commands to get info about physical/logical servers:

– by IP address:

```
select l_server_id from l_server_ips where l_server_ips = 'YOUR_IP_ADDRESS'
(enter your IP address)
```

– by logical server:

```
select p_server_id from l_server where id = 'YOUR__L_SERVER_ID'  
(enter your logical server id)
```

– selecting info about physical server:

```
select * from p_server where id = 'YOUR_P_SERVER_ID_'  
(enter your physical server id)
```

Creating Plan Wizards with XML

Introduction

Since version 2.4, H-Sphere supports XML-based plan wizards. Wizards are located in the `~cpanel/shiva/psoft/hsphere/plan/wizard/xml` directory. Location can be altered by setting `WIZARDS_DIR` to the current directory.

Editing the XML file would require proper knowledge of XML, so that the document remains well-formed and correct.

- [Adding new wizard to the list of plan wizards](#)
- [Defining plan wizard](#)

Adding New Wizard to the List of Plan Wizards

The list of wizards is defined in the `plan_wizards.xml` file. To add a new wizard, add the following line:

```
<wizard name="NAME" description="DESCRIPTION"/>
```

Here,

- NAME will be used as name of XML file without .xml extension (.xml suffix will be appended automatically). The file contains the plan wizard specification.
- DESCRIPTION is a lang string defined in hsphere_lang.properties. It should not contain "lang." prefix.

Defining Plan Wizard

New plan wizard definition starts from creating a new .xml file. The root XML element is:

```
<PlanWizard name="NAME" description="DESCRIPTION">
```

Attributes:

- NAME should match the filename (without .xml prefix) and NAME attribute in plan_wizards.xml file.
- DESCRIPTION is the plan wizard description from hsphere.properties file without lang. prefix.

PlanWizard element include the following entries:

```
<DefaultName>name</DefaultName>
```

This is the name that will serve as the default plan name in creating the plan.

```
<DefaultValues>
  <value name="NAME1">VALUE1</value>
  <value name="NAME2">VALUE2</value>
  ...
</DefaultValues>
```

Add default values to the plan. Usually, add `_template` and `menuId` values are added here.

```
<categories>
  <category>
  ...
  </category>
  <category description="DESCRIPTION">
  ...
  </category>
</categories>
```

categories tag is used to define resources. Resources are grouped into categories and described within the <category> element. Each category tag can have the description attribute which is optional. Categories are used in plan wizard screens to group resources into "logical" groups.

Following tags can be used inside category tag:

The <resource> element defines the resource class, name, description and include the following attributes if necessary:

- name – the name of the resource like account, mailbox;
- class – name of the resource class;
- help (optional) – description of the resource from hisphere_lang.properties file;
- unit (optional) – units for the resource, MB or GB;
- required (optional) – if it is set to 1 resource is required to be in the plan and regarded to be automatically included to the plan;
- include (optional) – if it is set to 1 resource will be marked as included by default to the plan wizard;
- active: (optional) – if active parameter is not set, no active checkbox will be available for the resource. Otherwise, if it is set to 1, resource will be marked as active by default. Any other value – active checkbox will be present, but not checked;
- noprice (optional) – if 1, the wizard will not prompt to enter pricing for the resource;
- ifresource (optional) – list of resources separated by the vertical bar '|' character. If any of the resources are included to or required in the plan, this resource will also be included.

Inside the <resource> element, <field> tag allows to get more info from the user for the specified resource. The data will be returned as a part of http request and can be used later via the #name parameter.

Attributes:

- name – name of the field, you can retrieve it later via the #name parameter;
- label – a label from hisphere_lang.properties, refers to the caption to be displayed in the wizard;
- type – type of the field to be displayed: textbox|checkbox.

If type = textbox, the following attributes are set:

- value – the default value;
- planvalue – name of the resource plan value (value defined in the plan); While editing the plan, the wizard will try to retrieve this value and show it as the textbox value;
- size – size of the textbox;
- check – yafv check (not implemented yet).

Example:

```
<field type="textbox" name="max_conn" label="planeditor.max_connections" value="10"
```

```
planvalue="MAX_CONN"  
size="4" check="vPriceReq" />
```

If type = checkbox, the following attributes are set:

- value – value of the checkbox;
- planvalue – name of the resource plan value (value defined in the plan). While editing the plan, the wizard will try to retrieve this value and, if it matches to the value, checkbox will be checked.

The <LogicalGroup> element defines a logical server, allows user to select one logical server if several are present.

Attributes:

- name – name of the group;
- type – type of the group;
- help – help message;
- default – default group.

Example:

```
<LogicalGroup name="unix_real" type="unix_real" help="admin-editwizard-o_lsgunix_real"/>
```

The <ifresource> element allows to group resources/LogicalGroup and activate them for the plan, only if the resource is enabled via global resources, or for the reseller plan.

Attributes:

- name – resource name. Will be checked by admin.isResourceDisabled;
- description – description of the resource from hsphere_lang.propeperities.

<ifgroup> element works the same way as ifresource but checks if the server group is available.

Attributes:

- name – server group name;
- else – resource name to show as unavailable.

The <resources> element is used to define resources and their initialization sequence.

Resources are defined in the resources element in the custom res_RESOURCE_NAME child tag where RESOURCE_NAME is the resource mnemonic identifier.

For example, for `unixuser` resource the tag would look like this:

```
<res_unixuser>  
</res_unixuser>
```

These tags contain the list `<mod>` tags where mods that will be defined in the plan are described.

Attributes:

- name – name of the mod; "" if missing;
- ifresource – list of resources separated with '|' sign. If any of those resources will be available in plan, i.e., required, selected through the web as included, or derived as included, the mod will be defined in the plan, otherwise it will not be defined.

Inside the mod element the following tags are structured:

The `<initresource>` tag is used to define `initresources` inside the mod.

Attributes:

- name – name of `initresource`. The system automatically checks that resource is included in the plan before adding `initresource`;
- ifactive – list of resources, separated by the '|' sign. If any of this resources are active in the plan (i.e, created by default), the `initresource` will be defined; otherwise, it will not be defined.

The `<initvalue>` tag is used to define `initvalues`.

Attributes:

- type – one of the following types as defined in `psoft.hsphere.plan.InitValue`:
`static`, `field`, `relative`, `absolute`, `relative_rec`, `absolute_rec`, `hostgroup`, `plan_free`,
`plan_value`
- label – is not used anymore, can be considered a comment.

The `initvalue` tag content is a string. If it starts with # sign, the value will be used as a name of the parameter which is passed via http request.

Here are several pre-defined variables that could serve as the `initvalue` tag content:

```
$STOPGAP_ZONE  
$STOPGAP_PREFIX  
$INSTANT_ZONE  
$INSTANT_PREFIX
```

Example:

```
<initvalue type="static" label="Home Directory">#homedir</initvalue>
<initvalue type="static">$INSTANT_PREFIX</initvalue>
```

The `<if>` tag allows include `initresources` or `initvalues` according to a certain condition. For example:

```
<if type="eq" left="#mixedip" right="dedicated">
  <true><initresource name="ip" mod="dedic_no_a"/></true>
  <false><initresource name="ip" mod="shard_no_a"/></false>
</if>
```

Here, the value of the `mixedip` parameter that is passed via HTTP request is checked for being equal to the value of the `dedicated` parameter.

The `<values>` element included `<value>` tags inside.

The `<value>` tag used to define the resource values in the plan.

Attributes:

- `name` – mnemonic identifier. If the tag content string starts with # sign, the value will be used as a name of the parameter that is passed via HTTP request.

Example:

```
<value name="SSI">1</value>
```

The `<special>` element is used to define some additional settings such as tld pricing. It allows to add checkbox to any resource and, if checkbox is selected, to include another template.

To define the *special* attribute for resource, create the `<res_RESOURCE_NAME>` tag (like `<res_opensrs>`) inside the `<special>` element (you can have multiple tags inside special section).

Inside `<res_RESOURCENAME>` tag, the `<field>` tag is used to define a field:

Attributes:

- `type` – checkbox type;
- `name` – name of the checkbox field;

- label – label from `hsphere_lang.properties` referring to the field caption;
- value – value of the checkbox;
- checked – equals 1 if checked.

Inside the `<field>` element, `<include>` tag defines a template to include:

Attributes:

- ifvalue – a value to match against the field; if it matches, the template is included;
- name – the name of the template to be included.

Example:

```
<special>
  <res_opensrs>
    <field type="checkbox" name="leave_osrs_prices"
      label="planwizard.leave_osrs_prices" value="1" checked="1">
      <include ifvalue="" name="admin/wizards/tldprices.html"/>
    </field>
  </res_opensrs>
</special>
```