H-Sphere Developer Guide

PDF document generated: 04 Jan 2008

Welcome to H-Sphere Developer guide. It gives a look inside H-Sphere and explains how to perform development tasks.

Getting User Information from the H-Sphere System Database

To get user-specific information from the system database:

1. *Start PostgreSQL* on CP server.
2. *Log into the H-Sphere system database*
3. Run below queries to get the following info:
    - User Info
    - Mail Accounts Info
    - Resellers Info
    - Credit Card Info
    - Selecting Reseller's Users Credit Card Info:
    - Credit Card Types
    - 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 Email Address
    - All Users' Emails but Resellers'
    - User Emails by Their MySQL DB Names
    - Verify Users
    - Logical and Physical Servers
    - User Billing Info
    - Billing Periods
    - Payment Modes
    - Plan Prices
    - Domain Registration Prices

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 reseller's users credit card info:

This is possible if you don't encrypt credit card numbers in H-Sphere database. In this case run the following query:

```
select cc.cc_number, cc.name, cc.exp_year, cc.exp_month, cc.type from accounts as acc join credit_card as cc on
(cc.id=acc.bi_id) where acc.reseller_id=RESELLER_ACCOUNT_ID;
```

Make sure to replace RESELLER_ACCOUNT_ID with the this reseller's account ID.

Selecting credit card types:

a) All supported credit cards: `select * from cc_brands;`

b) Credit cards allowed in Merchant Gateway: `select * from active_merch_gateway;`

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 emails of all users but resellers':

To receive the list of all email addresses ignoring resellers' end users, run the following query:
`select c_i.email from contact_info c_i join accounts a on (c_i.id=a.ci_id and a.reseller_id=1);`

Getting user emails by their MySQL DB names

```
select c.email from mysqldb m, parent_child pc, accounts a, contact_info c where m.db_name='DBNAME' and m.id=pc.child_id and pc.child_type=6001 and pc.account_id=a.id and a.ci_id=c.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)

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);

Billing periods for plans:

a) All information including discounts if present:
```
select * from plan_value where name like '_PERIOD%' and plan_id=plan_id;
```
b) Discounts:
```
select * from plan_value where name like '%_DISC_%' and plan_id=plan_id;
```


Payment modes:

```
select billing from plans where id=plan_id;
```

`plans.billing` takes the following values:

- `0` - without billing;
- `1` - billing mode;
- `2` - billing with trial period.

Plan pricing details:

```
select * from plan_value where plan_id=plan_id;
```

Domain registration pricing details:

a) `select * from tld_prices;`

b) `select * from plan_value where name like '_TLD_%';`

Database Connections

Related Docs: · Getting Information from the H-Sphere Database

There are two possible types of connections to the H-Sphere database:

- Regular shared connections
- Connections with opened transaction

IMPORTANT:
In any case, you should use only **prepared statements** when working with the H-Sphere database!

Regular Shared Connections

The part of the code which establishes and uses a regular database connection should look something similar to this:

```
PreparedStatement ps = null;
Connection con = Session.getDb();
try {
    ps = con.prepareStatement(...........);
    ....................................
} finally {
    Session.closeStatement(ps);
    con.close();
}
```

Connections With Opened Transaction

In general, the part of the code which establishes and uses a transaction connection should look as follows:

```
// checking if the transaction exists
boolean wasTrans = Session.isTransConnection();
```



```
Connection con = wasTrans ? Session.getDb() : Session.getTransConnection();
try {
    // some operation with the database
} catch (Exception ex){
    if (!wasTrans) {
        // rollback transaction
        con.rollback();
        // here should also be some data to release cache and to synchronize with old data in HS DB
    }
    throw ex;
} finally {
    if (!wasTrans) {
        // commit transaction and release connection
        Session.commitTransConnection(con);
    } else {
        con.close();
    }
}
```

Important:

1. There is the pool of transaction connections (5 by default). Opening a new transaction that exceeds this maximum would cause the system to hang up.
2. As the number of available transactions is resctricted, you must release the transaction connection you have opened by using the following command:

   Session.commitTransConnection(con);
3. To get an opened transaction connection:

   con = Session.getTransConnection();

   Later in the code, you may get the same connection:

   con = Session.getDb();

   Never close a transaction connection you haven't opened!
4. Please make sure you synchronize your objects with the corresponding data in the database. You must release your cache if you make a rollback.

Related Docs: · Getting Information from the H-Sphere Database

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 -Xrunjdwp: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
Debuger Mode: Attach
Host: localhost
Port: 9999
```

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

TimeUtils Class

The `TimeUtils` class is designed to subsitute all other H-Sphere classes dealing with time. This class is needed for testing any H-Sphere features that use time (like accounts, domain registration renewals, and the like).

From now on, please **ALWAYS** use the following TimeUtils wrappers instead of any other Date/Time functions:

```
System.currentTimeMillis() = TimeUtils.currentTimeMillis()
new java.util.Date = TimeUtils.getDate()
Calendar.getInstance() = TimeUtils.getCalendar()
Thread.sleep(s) + TimeUtils.sleep(s)
```

Other utilities **HIGHLY** recommendable for usage:

```
java.sql.Date getSQLDate()
java.sql.Timestamp getSQLTimestamp()
java.sql.Time getSQLTime()
java.util.Time getTime()
```


External Credits

Related Docs: · CP Cron Configuration

H-Sphere supports external credits, or external charges: charges exported from external billing systems.

External charges are stored in the `external_credits` table. The ecCron internal CP cron adds charges from external billing software to the `external_credits` table as the accounts' credits.

To set the time interval when ecCron is launched, you need to add the following line to hsphere.properties:

```
EX_CHARGE_CRON=5
```

In the above example, ecCron would start every 5 minutes.

The `external_credits` table is created during H-Sphere upgrade and has the following structure:

```
CREATE TABLE external_credits (
   id int NOT NULL,
   account_id int4,
   amount float,
   created timestamp with time zone,
   description varchar(128),
   PRIMARY KEY(id)
);
```

Related Docs: · CP Cron Configuration

Account Preferences

You can set individual interface preferences: preferred design (skin), icon set, or language - for a particular account.

This can be implemented within an H-Sphere *template* by using the following Freemarker commands:

- `<assign res = account.preferences("KEY", "VALUE")>` - set account preferences.

  Here:

  `KEY` is the key of an interface preference to be assigned to an account (up to 64 chars),
  `VALUE` is its value (up to 256 chars).

  The following keys are available in H-Sphere (refer to the *Skin And Icon Set Customization* document in Customization Guide for detailed description of the `~cpanel/shiva/psoft/hsphere/design_config.xml` file):

  `design_id` - preferred design id (in `design_config.xml`);
  `icon_image_set` - preferred icons image set (in `design_config.xml`);
  `skill_icon_set` - preferred icons skill set (in `design_config.xml`);
  `lang` - preferred language.

  For example, to assign the XPressia design for an account:

  ```
  <assign res = account.preferences("design_id", "xcp")>
  ```

- `<assign property1 = account.preferences("KEY")>` - retrieve account preferences.

Script Runner

Script-runner is a utility that launches H-Sphere scripts on Unix/Linux servers.

H-Sphere control panel establishes an SSH connection with a Unix box and runs script-runner:

```
ssh -a -x root@xxx.xxx.xxx.xxx /hsphere/shared/scripts/script-runner.pl
```

Further, the control panel sends commands to the output stream of the SSH session and script-runner reads and performs the commands and then outputs.

The format of command passed to script-runner is:

```
COMMAND=command_name&PARAM1=param1&..&PARAMN=paramn&INPUT=input\n
```

It starts with the keyword `COMMAND` followed by actual command. Then, if necessary, it's followed by the parameters that are numbered from 1 and further. All parts of the command are separated with the ampersand (&). The '=' symbol is followed by the value. The command ends with the line feed character.

If the command requires standard input, you should set the `INPUT` parameter. Commands are passed in the 8-bit ASCII encoding. Command parameters and standard input are passed in the URL-encoding. Version 1.8 contains script-runner-b64 utility that has the same funcionality as script-runner, but data should be passed in the Base64 encoding and it returns the result in the same encoding.

Script-runner returns output in the following format:

```
OUTPUT=some_output&ERROR=text_of_error_or_empty_string&EXIT_CODE=0
```

If there is an error during script execution, parameter `ERROR` will contain the error text and output code `EXIT_CODE` will be nonzero. OUTPUT is a standard script output.

To check if script-runner is ready to accept commands, use 'PING' command. You should get the 'PONG' output.

Script-runner 1.8 logs all executed scripts and their output. They can be found in the `/var/log/scriptrunner` directory on the box where script-runner is running.

H-Sphere XML API Reference

What Is H-Sphere XML API

H-Sphere XML API, written in Java, is designed to manage H-Sphere services from remote applications via SOAP. It uses the Apache Axis implementation of SOAP and runs under the *Apache Tomcat* engine.

Remote applications interact with CP services by means of XML-formatted requests transferred via HTTP (direct TCP connection can also be set up). Error messages are also processed in XML.

XML schema is based on the SOAP RPC convention.

Mechanism of Interaction

XML API provides the following mechanism of communication between remote applications and the Control Panel:

- A remote Java application calls CP methods to perform particular actions (for example, to create an account or to return the list of domains). For this, it invokes its SOAP client to form the corresponding XML request and to http it to the Control Panel;
- the CP SOAP server receives this XML request, parses the request> and calls a target H-Sphere service;
- the target H-Sphere service performs actions according to the received request (for example, creates an account or returns the list of domains), and responds via the SOAP server to the remote application.
  If the request is incorrect, the SOAP server returns XML with SOAP fault.

Therefore, though HTTP is used for data transfer, **H-Sphere XML API services are not called from a Web browser**!

Where To Get H-Sphere XML API

For Java, you may download H-Sphere XML API from psoft.net:

- Axis libraries
- H-Sphere Java XML API library
- Sample Java classes for H-Sphere versions 2.5 and 3.0+ for testing H-Sphere XML API functionality

Downloads correspond to the *latest H-Sphere version under development*.

For other programming languages (for Java as well), you can use WSDL file to implement H-Sphere XML API services in this language. See how to generate WSDL via Axis.

Here you may find a sample H-Sphere signup form written in PHP 5 (which has an embedded SOAP client) using H-Sphere XML API.

How To Enable And Test H-Sphere XML API

Please read a separate document on enabling and testing H-Sphere XML API.

Services

Actions supported by XML API are grouped by services:

- Admin Services - basic admin services: adding new accounts, etc.
- User Services - basic user services: getting account information
- Domain Services - creating/deleting domains and subdomains
- Web Services - managing Web options
- FTP Services - managing FTP options
- User FTP Services - managing user FTP resources: subusers, passwors, etc.
- DNS Services - managing DNS
- Mail Services - creating/deleting mailboxes, mailing lists, autoresponders, mail forwards, etc.

- MySQL Services - managing MySQL databases and users
- PostgreSQL Services - managing PostgreSQL databases and users
- MSSQL Services - managing MSSQL databases and users
- Support Services - managing trouble tickets (H-Sphere support center)
- Migration Services - migrating users and resellers by means of corresponding XML forms
- PGP Services - PGP encryption/decryption in H-Sphere trouble tickets

See the up-to-date *XML API services documentation* generated by Javadoc tool.

NOTE: When generating `AuthToken` for reseller, in addition to defining reseller's accountID, login and password, set also login in the Role to `""` (empty) and account id in the Role to `"0"`.

Enabling H-Sphere XML API

Related Docs: · H-Sphere XML API Reference · H-Sphere XML API Security Settings

H-Sphere XML API is designed to manage H-Sphere services via SOAP protocol. It is based on Apache Axis implementation of SOAP and runs under *Apache Tomcat engine*.

You can download Axis libraries and the latest version of XML API libraries from PSoft site.

WARNING:
H-Sphere XML API is an experimental feature. Please be careful in using it for critical tasks!

Enabling XML API

To enable XML API in H-Sphere:

1. Log into the CP server as the *cpanel user*.
2. **H-Sphere 2.4.3 Patch 11 and up:** Edit the `~cpanel/shiva/psoft_config/allow_access.properties` file to grant access to CP server via SOAP port from external IPs.

   [-] SOAP access in earlier H-Sphere versions

   Important: In H-Sphere before 2.4.3 Patch 11 SOAP access to CP server via standard 8080 port was provided by default.
   The following procedure explains how to configure SOAP access via port 8180 and should not be performed in H-Sphere 2.4.3 Patch 11 and up!

   1. Copy the wsdd config file:
      `cp ~cpanel/shiva/psoft/hsphere/axis/server-config.wsdd ~cpanel/hsphere/WEB-INF/`
      Skip this step if the file is already in the target location.

2. Edit the Tomcat XML configuration file:
   `vi ~cpanel/jakarta/conf/server.xml`
3. Find the following commented fragment:

```
<!--Connector port="8080"
           maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
           enableLookups="false" redirectPort="8443" acceptCount="100"
           debug="0" connectionTimeout="20000"
           disableUploadTimeout="true" /-->
```

   and replace it with the following code:

```
<Connector port="8180"
        maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
        enableLookups="false" acceptCount="100"
        debug="0" connectionTimeout="20000"
        disableUploadTimeout="true" />
```

   Optionally, leave the port at 8080.
3. *Restart H-Sphere*.

After that, XML API will be available. You may wish to test if it works correctly by means of our test Java classes that use H-Sphere XML API.

Testing HS XML API

1. Make sure you are logged into the CP server as the *cpanel user*.
2. Download the latest version of test Java classes from PSoft site to the `~cpanel/shiva/psoft/hsphere/axis/test` directory (create it first if it does not exist):
   ```
   cd ~cpanel/shiva/psoft/hsphere/axis/test/
   wget http://www.psoft.net/shiv/HS/xml-api/AxisExamples.tgz
   ```
   Note: AxisExamples.tgz is for the latest H-Sphere 3.0 version. For H-Sphere 2.5, download AxisExamples_25.tgz.
3. Extract the archive with the command:
   `tar zxf AxisExamples.tgz`
   Java source `.java` files will be added into the current `~cpanel/shiva/psoft/hsphere/axis/test/` directory.

4. Edit the necessary test classes (at least `AdminServicesTest.java`) replacing URL in the folowing line:
   `String endpoint = "http://localhost:8080/psoft/servlet/AdminServices.jws";`
   with your URL:
   `String endpoint = "http://your_cp_host:8180/your_cp_url_path/AdminServices.jws";`
   where `your_cp_url_path` is by default `psoft/servlet` but you may change it in your *Tomcat configuration*.
   If you have enabled XML API with port 8180, make sure it is specified in the URL.

   Note: don't forget to include XML API libraries into your java classpass

5. Compile Java classes:
   cd ~cpanel/shiva/psoft/hsphere/axis/test/
   javac *.java
6. Upon the successful compilation, run the following command:
   java psoft.hsphere.axis.test.AdminServicesTest admin <hs_admin_password>
7. You will see the > prompt. Here, enter the line:
   `searchbydomainname` domain-name
   where domain-name is a customer's domain name.
8. On the next > invitation, press Enter to force the service to run.
9. Repeat the previous two steps testing other services, for instance `getAccounts`, `getDomains`, `searchbydomainname`, etc:
   [cpanel@exampleuser test]$ java psoft.hsphere.axis.test.AdminServicesTest admin admin
   >getAccounts example.com
   >Params:1:example.com
   1

   >getDomains example.com
   >Params:1:example.com
   no domains
   See in H-Sphere XML API Reference to learn how to invoke other Web services via XML interface.
10. Press CTRL-D to stop.

Getting the List of Available Services

You may get the list of available XML services by typing in your browser:

`http://your_cp_host:8180/your_cp_url_path/AxisServlet`
where:
`your_cp_host` is your CP host domain (`CP_HOST` in hsphere.properties), and
`your_cp_url_path` is usually `psoft/servlet` if not configured otherwise in your *Tomcat configuration*.

If this doesn't work, check servlet mapping in your Tomcat configuration:

1. As the *cpanel user*, open the `~cpanel/hsphere/WEB-INF/web.xml` file for edition:
   `vi ~cpanel/shiva/psoft/hsphere/WEB-INF/web.xml`
2. Make sure the following lines are present in the file:
   ```
   <servlet>
   <servlet-name>AxisServlet</servlet-name>
   <display-name>Apache-Axis Servlet</display-name>
   <servlet-class>
   org.apache.axis.transport.http.AxisServlet
   </servlet-class>
   </servlet>
   ```
3. Find the following mapping block:
   ```
   <servlet-mapping>
   <servlet-name>AxisServlet</servlet-name>
   <url-pattern>*.jws</url-pattern>
   </servlet-mapping>
   ```
   and add the following lines right underneath:
   ```
   <servlet-mapping>
   <servlet-name>AxisServlet</servlet-name>
   <url-pattern>/AxisServlet</url-pattern>
   </servlet-mapping>
   ```
4. *Restart H-Sphere*

5. Try again this URL in your browser:


```
http://your_cp_host:8180/your_cp_url_path/AxisServlet
```

You may also wish to test AxisTestServlet from the package of test Java classes:

```
http://your_cp_host:8180/your_cp_url_path/AxisTestServlet
```

Generating WSDL

You can generate WSDL files for H-Sphere XML API services by means of the Axis' `org.apache.axis.wsdl.Java2WSDL` utility.

For example, to generate a WSDL file for `SupportServices`, log into CP server as cpanel and run:

```
java org.apache.axis.wsdl.Java2WSDL -o SupportService.wsdl -l"http://localhost:8080/psoft/servlet/AxisServlet"
psoft.hsphere.axis.SupportServices
```

Example: H-Sphere XML API signup form implementation in PHP 5 (SOAP client embedded) via WSDL.

Related Docs: · H-Sphere XML API Reference · H-Sphere XML API Security Settings

FreeMarker Access Control List (FMACL) Objects

FreeMarker Access Control List (FMACL) Objects provide an easy and effective way to manage H-Sphere resources and grant different levels of access to their methods from FreeMarker templates. This technology is especially useful for making changes in third-party products implemented as *H-Sphere packages* (e.g., custom plans and reports).

Purpose

Formerly, when we needed to handle the same functionality related to different objects (take domain registration as an example), we would split the code among these resources. For example, some of the methods would be in BillViewer, some in BillManager, CreateUser, HsphereToolbox, etc... This led to problems with changing the code - it had to be changed across a wide variety of classes. Also, due to that, the same code was often repeated.

Especially, it was hard for third parties to integrate such functionality to H-Sphere. In particular, to add a custom resource manager, the resource should be added and should be represented in all admin plans, which, in turn, required changes in the system database.

The idea comes from the fact that some functionality is not really a "resource", but a set of facilities of common purpose (for example, report generation), accessed differently on different levels. This solution is implemented in FMACL objects.

Implementation

To provide access to required methods, the new FreeMarker `obj` tag is introduced. Now the methods will be accessible in FreeMarker templates as `obj.`key.method. For example, to call the `getTLDPrice` method of the `psoft.hsphere.admin.DomainRegistration` class from a template, we simply write:

```
obj.domreg.getTLDPrice()
```

Here, `domreg` is a key associated with `psoft.hsphere.admin.DomainRegistration` in the `acl_objects.xml` file.

Each object is instantiated via default constructor (without params) and must implement the FreeMarker's `TemplateHashModel`.

Access permissions to class methods must be set in the corresponding `.acl` files for each class, located in the same directory and bearing the same name as the class

files. For example, access permissions for the `psoft.hsphere.admin.DomainRegistration` class must be set in the `psoft/hsphere/admin/DomainRegistration.acl` file.

FMACL Objects XML File

The objects are defined in **`~cpanel/shiva/psoft/hsphere/acl_objects.xml`** in the following format:

```
<objects>
<object key="reports" class="psoft.hsphere.reports.ReportManager">
<object key="domreg" class="psoft.hsphere.admin.DomainRegistration">
...
</objects>
```

Here, each unique `key` corresponds to an H-Sphere class. Each class defined here must have its `.acl` file in the same directory and with the same name. For example, for `psoft.hsphere.admin.DomainRegistration` there must be the `psoft/hsphere/admin/DomainRegistration.acl` file with access permissions for used methods of this class. See more on setting levels of access to class methods.

To re-define or customize (merge) the default `acl_objects.xml` file, set the `ACL_OBJECTS` property in a package properties file. See *Customizing XMLs With Packages* for details.

Setting Levels of Access to Class Methods

Access permissions to class methods are set in hsphere_class`.acl` files in the following lines:

```
key access_mask
```

where:

- `key` is template hash model key;
- `access_mask` is a combination of the following characters defining the level of access:
  a - H-Sphere admin
  r - reseller admin

u - user
e - everyone, including access from the outside of H-Sphere.

Access mask specifies on which level a key is accessible (the level will be determined by checking the plan of a user logged in). If the key is not accessible, the system will throw TemplateModelException.

Example for setting access permission for domain registration (`domreg`):

```
getTLDPrice aru
setTLDPrice ar
lookup e
enableTLD a
```

This means that the `getTLDPrice` method will be accessible by admin, reseller admin and user, `setTLDPrice` by admin and reseller admin, `lookup` by everyone (even if user is not logged in), `enableTLD` only by admin.

The access will be provided by calling `obj.domreg.lookup`, `obj.domreg.geTLDPrice`, etc., in H-Sphere templates.

Global Resources And Resource Dependencies

(version 2.5 and higher)

In version 2.5, the working scheme for global resources in H-Sphere was improved. To simplify describing and provide flexible access mechanism, it was logically split into two parts: 'globals' and 'resource dependencies'.

'Globals' is the mechanism to provide enabling/disabling some features (resources) both for the entire system and for separate reseller hosting systems. By now, we have the following Globals: H-Sphere resources, "key - value" like objects stored in db, Class managed key sets, Class maintained sophisticated objects (former "pseudo resources").

'Resource Dependencies' is a description of how resources/objects depend on other resources/objects. Not only chains of dependencies up to Globals can be defined here, there is also a possibility to describe parent-child and ancestor-descendant resource dependencies. Such resource dependencies are used to make a decision which resources can be added to user accounts and which cannot.

globals.xml

This file lists global resources. Location: `~cpanel/shiva/psoft/hsphere/globals.xml`
Example of globals.xml

DTD scheme:

```
<!DOCTYPE globals [
<!ELEMENT globals (section+, special?)>
<!ELEMENT section ((object | plan_dependent_resource | set)*)>
<!ELEMENT object (check_config?)>
<!ELEMENT plan_dependent_resource (check_config?, user_plan*)>
<!ELEMENT set (check_config?)>
<!ELEMENT maintained_object EMPTY>
<!ELEMENT check_config (property+ | variant+)>
```



```
<!ELEMENT variant (property+)>
<!ELEMENT property EMPTY>
<!ELEMENT special (maintained_object+)>
<!ELEMENT user_plan EMPTY>

<!ATTLIST section id CDATA #REQUIRED>
<!ATTLIST section label CDATA #IMPLIED>
<!ATTLIST section description CDATA #IMPLIED>
<!ATTLIST section show (each | globals | plans | custom) "each">
<!ATTLIST section store (each | settings | plans | custom) "each">
<!ATTLIST section online_help CDATA #IMPLIED>
<!ATTLIST object name CDATA #REQUIRED>
<!ATTLIST object label CDATA #IMPLIED>
<!ATTLIST object label_enabled CDATA #IMPLIED>
<!ATTLIST object label_disabled CDATA #IMPLIED>
<!ATTLIST object default (enabled | disabled | unavailable) "enabled">
<!ATTLIST plan_dependent_resource name CDATA #REQUIRED>
<!ATTLIST set name CDATA #REQUIRED>
<!ATTLIST set label CDATA #REQUIRED>
<!ATTLIST set default (enabled | disabled | unavailable) "enabled">
<!ATTLIST set class CDATA #REQUIRED>
<!ATTLIST set managed (globally | reseller) "globally">
<!ATTLIST property key CDATA #REQUIRED>
<!ATTLIST property value CDATA "*">
<!ATTLIST maintained_object name CDATA #REQUIRED>
<!ATTLIST maintained_object class CDATA #REQUIRED>
<!ATTLIST user_plan type CDATA #REQUIRED>
<!ATTLIST user_plan label CDATA #IMPLIED>
<!ATTLIST user_plan default (enabled | disabled | unavailable) "enabled">
]
```

It can be seen that the root level consists of one or more 'section' elements. Section is the way to combine several 'global' objects or sets into a group and to show them in H-Sphere CP interface in a suitable form.

Beside this, there is also one more auxiliary section 'special' which is used to define "Class Maintained Global Objects".

Elements and attributes:

- section:
  - id: mnemonic identifier; mandatory attribute, the rest are non-mandatory
  - label: a key having correspondences in the language bundle files

- description: a key, like the above one, for the text describing some peculiarities of the section
- online_help: a key having correspondence in the online_help.xml file. You may choose which one of 'description' or 'online_help' is more suitable in each particular case.
- show: by default is set to show always. If necessary, you may restrict this rule by specifying one of the following values:
  - globals: to show in master admin's 'globals' menu
  - plans: to show in reseller plan wizards only
  - invisible: to exclude the section from standard page flow and show it somewhere else on the page calling it by hand (e.g.: admin.getGlobalSection)
- store: by default, it is set to "each" what means storing the "status" values of all objects in the section both into the 'settings' table of the "hsphere" database and in the table 'plan_value" for reseller plans. Normally, the attributes 'show' and 'store' should match to each other (have the similar values). Be careful with changing these attributes as it may lead to undesired results. For instance, if the attribute 'store' is set to "settings", whereas the section objects are shown in reseller plan wizards, these values won't be stored in the 'plan_value' table, so they won't affect separate reseller hosting systems. And one more case, if the attribute 'store' is set by default to "each", but the globals from this section are prohibited to be shown on the page 'Globals' (e.g. attribute 'show' is set to "plans"), the respective values will be set to "DISABLED" in the 'settings' table. So, all the object in the section will become "Disabled".

- 'Global' object: an H-Sphere resource or a simple "property" that has a mandatory attribute 'name' (treat this as a property key). An example of 'global' object is 'cp_ssl_ip_based'. By default, all global objects are "Enabled" unless you decide to change this by specifying the value explicitly (attribute 'default').
  - default: if necessary, you may change it to 'disabled'
  - name: mandatory attribute, the rest attributes are non-mandatory
  - label: the same as for 'section'
  - label_enabled, label_disabled: replace the 'label' attribute and are used to show the object as radio button instead of checkbox
- plan_dependent_resource (NEW! in H-Sphere 3.1): added to provide solution for disabling resources for different plan types separately, frontpage for Unix and Windows platforms in particular
- set of keys: the way to describe a group of entities where each of them contains a unique key, description and status. An example of a set is "Logical Server Groups". Each server group has a key which is group id, description and status. Due to significant difficulties to describe each global set in XML, the behavior of this type of 'globals' is realized by a Java class specified with attribute 'class'. The other attributes 'label' and 'default' are described above.
  - managed: this attribute shows if this global resource, being managed by admin, is also manageable by reseller: managed="reseller". In particular, this is implemented in *dedicated server template* where resellers can set prices for templates provided by main hosting provider (admin).
    If managed is not set, it is considered to be "globally" by default (i.e., not manageable by resellers).
- maintained_object: "Class Maintained Global Objects" belong to **special** type of global objects. Each object is **maintained** by a java class specified in the attribute 'class' in the xml. To get a "Maintained Global Objects" working, all you need is to write a **maintainer** class **maintainer**. This class should **extend** standard class 'psoft.hsphere.global.ClassMaintainedGlobalObject?'. In your class you either
  (a) override the method boolean isEnabled() or
  (b) override all the 3 methods: boolean isEnabled(Reseller r), boolean isEnabled(int resellerPlanId), boolean isEnabled(ServletRequest rq) or
  (c) override all the methods: Globals.State getObjectState(Reseller r), Globals.State getObjectState(int resellerPlanId) and Globals.State getObjectState(ServletRequest rq).

Doing (a) and (b), you can have only 2 possible **statuses** for the objects: ENABLED or UNAVAILABLE. The method (c) allows you to have any status you need (at least those ones defined in class Globals.State). Please note that the "Maintained Global Objects" are neither shown on 'Globals' page in the CP interface nor they are shown in Reseller plans. Their states cannot be stored in the database either. Besides, none of the true "H-Sphere resources" can be made "Maintained Global Object".

- check_config: each of the objects/sets described above may require additional checks whether it is configured to work properly in the `~cpanel/shiva/psoft_config/hsphere.properties` file. To provide this, the additional structure is added:

```
<check_config>
      <property key="..."/>
      <property key="..."/>
      ....
</check_config>
```

Here, an object will be considered as "configured" if all of the specified properties are defined in the `hsphere.properties` file, no matter what values they have. You may strengthening this by specifying the desired values in the 'property' elements (attribute 'value').

If necessary, you may also implement the "OR" logic in conditions by including the following structure::

```
<check_config>
    <!-- Variant 1 -->
    <variant>
        <property key="..."/>
        <property key="..."/>
        ....
    </variant>
    <!-- Variant 2 -->
    <variant>
        <property key="..."/>
        <property key="..."/>
        ....
    </variant>
...
</check_config>
```

This means that an object has either Variant 1 of **each** of the specified properties set, or Variant 2 with its list of properties set, etc.

Note: The same scheme works for describing resource dependencies in `resource_dependences.xml` (see below). For example:

```
<resource name="hosting">
    <variant>
```



```
            <requires name="domain" relation="parent"/>
        </variant>
        <variant>
            <requires name="parked_domain" relation="parent"/>
        </variant>
        <variant>
            <requires name="3ldomain" relation="parent"/>
        </variant>
    </resource>
```

Here, the `hosting` resource can be a child resource for either `domain` (Transfered Domain), or `3ldomain` (Third-Level Domain), or `parked_domain` (Parked Domain) resources.

resource_dependences.xml

This file describes relationships between parent and dependent resources. Location: `~cpanel/shiva/psoft/hsphere/resource_dependences.xml`
Example of resource_dependences.xml

This xml file is based on the following DTD:

```
<!DOCTYPE dependences [
  <!ELEMENT dependences (resource+)>
  <!ELEMENT resource (requires+ | variant+)>
  <!ELEMENT variant (requires+)>
  <!ELEMENT requires EMPTY>

  <!ATTLIST resource name CDATA #REQUIRED>
  <!ATTLIST requires name CDATA #REQUIRED>
  <!ATTLIST requires relation (parent) #IMPLIED>
]>
```

As you can see, this is quite a simple structure, so it is very easy to write resource dependence rules. E.g. to become available, resource 'vps_ip' requires enabling resource 'vps':

```
<resource name="cf_dsn_record">
    <requires name="odbc"/>
    <requires name="cf"/>
```



```
</resource>
```

Using the attribute 'relation' you may specify that the required resource is a direct **parent** or **ancestor** for the current resource.

List of Resource Dependencies

Note: This table of resource dependencies is listed here as an example for H-Sphere 2.5 Beta 1. It will be modified with subsequent H-Sphere versions.

Dependent Resource	Parent Resource(s)
miva	empresa
oscommerce	MySQL
miva_lic_manager	miva
urchin_lic_manager	urchin
MySQLDatabase	MySQL
MySQLUser	MySQL
mysqldb_quota	MySQL
MSSQLLogin	MSSQL
MSSQLDatabase	MSSQL
MSSQLUser	MSSQL
MSSQLQuota	MSSQL
pgsqldatabase	pgsql
pgsqluser	pgsql
pgsqldb_quota	pgsql
real_user	realserver_user

real_server_traffic	realserver_user
winquota	realserver_user
cfentry	cf
mnogosearch	MySQL
phpbb	php, MySQL
asp_secured	asp
sshresource	sshmanager
vps_ip	vps
vps_mem_limit	vps
vps_proc_limit	vps
vps_ip_traffic	vps
kanoodlemanager	kanoodle
ds_custom_build	ds_enable
ds_manager	ds_enable
ds_reboot_enable	ds_enable
ds_backup_enable	ds_enable
allow_ds_resell	ds_enable
allow_own_ds	ds_enable
ds	ds_enable
ds_bandwidth	ds
cf_dsn_record	odbc, cf

Building Packages

Related Docs: · Template Customization With Packages · XML Customization With Packages · Building Language Packages

This document explains what are packages in H-Sphere and how to build them.

Overview

H-Sphere offers an interface for integrating custom plugins, or packages. With these packages, you may add new or override default H-Sphere functionality.

Packages can be used to integrate the following elements:

- resources: new or custom Java classes
- system scripts (in the `/hsphere/shared/scripts` directory)
- templates: new or custom templates
- images: custom images
- *Control Panel menu*
- *CP skins and icon sets*
- *language bundles*: adding new languages to H-Sphere interface or updating language files

- *context help XML customization*
- plans: XML plan wizards
- crons: new or custom CP cron jobs
- promotions: custom promotion validators and calculators
- SQL quieries: making changes into the H-Sphere database
- third party RPMs and tarballs to be included into H-Sphere scripts.

These components are built into one portable `.hsp` file which can be easily *installed* by an H-Sphere system administrator.

Building Procedure

Building packages is performed in the following steps:

1. **preconfiguration**:
   - determine package configuration: what resources, templates, systems scripts, XML configuration files, language bundles, jars, etc. would be included into a package
   - prepare package content: make sure the files to be included to a package are located in the corresponding directories you will specify in the package configurator's command line
   - run package configurator to collect package source files into one preconfigured package source directory
2. **configuration**:
   - set package info: package name, version, build, etc.
   - set package custom properties configuration: custom XML configs, language bundles, etc.
   - check and manually add source files wherever needed
3. **building**:
   - assemble the package into one Java archive file with `.hsp` extension

1. Preconfiguration

1. Log into the CP server as *cpanel user*.
2. Run **package configurator** to prepare the structure of your package. The format is:

java psoft.hsp.tools.PkgConfigurator --with-prefix[=TARGET_DIR] [WITH-OPTIONS] [-l LOG_FILE [-dl]]

Here:

- `TARGET_DIR` is the directory where the package will be assembled. Package configurator creates the directory structure where you will later add files to build the package. If the specified directory does not exist, the configurator will try to create it. If `TARGET_DIR` is omitted, the directory structure will be created in the current directory.
- `WITH-OPTIONS`: you choose the options to specify which components you will include into the package: templates, language bundles, images, scripts, etc. See Package Tools for details.
- `LOG_FILE`: with the `-l` option, package configurator writes its actions to a log file where you could check if it worked through correctly; `-dl` is a detailed log mode.

Example:

```
cd ~cpanel/shiva/custom/Packages
java psoft.hsp.tools.PkgConfigurator --with-prefix=./MyPackage --with-properties --with-templates
--with-lang-bundle --with-classes --with-scripts --with-xmls -l conf.log -dl
```

Notes on usage:

1) It is convenient to create a special location where you will assemble your packages. For example, you can create the `~cpanel/shiva/custom/Packages` directory and specify the `MyPackage` subdirectory as `TARGET_DIR` for package configurator.

2) Normally, you don't need to specify paths in `WITH-OPTIONS`. In such case, package configurator will create empty directories (with empty default files for some options) within the `TARGET_DIR` directory, and you will be able to add the files later.

Inside the package directory (specified in the `--with-prefix` option), package configurator creates:

- the **package configuration file** `_pkg.xml`. In this XML file you define the package info (name, version, build, vendor, description) and configure what elements will be included into the package.
- the `src` directory where all source files are collected in the following directories and files specified in `_pkg.xml`, in accordance with the configurator's options chosen:

  - `templates/` - custom templates;
  - `pkg_classes/` - custom Java classes;
  - `pkg_scripts/` - custom scripts;
  - `pkg_xmls/` - custom XML configuration files;

- `pkg_jars/` - 3rd party JARs;
- `pkg_config/` - custom package properties (the empty `default.properties` file created);
- `pkg_lang_bundle/` - language bundles
- `pkg_images/` - custom images
- `pkg_tarballs` - 3rd-party tarballs
- `pkg_rpms` - 3rd-party RPMs
- `_SCRIPTS/_pkg.sql` - SQL queries to be performed upon the H-Sphere database
- `_SCRIPTS/_pre-install` - script to be executed before the *package installation*
- `_SCRIPTS/_post-install` - script to be executed after the package installation
- `_SCRIPTS/_pre-uninstall` - script to be executed before the *package uninstallation*
- `_SCRIPTS/_post_uninstall` - script to be executed after the package uninstallation

2. Configuration

Package Configuration File (_pkg.xml)

DTD Scheme | Description of Tags and Attributes in _pkg.xml | Example

According to the options selected, package configurator creates the package configuration file `_pkg.xml` which looks similar to this:

```
<?xml version="1.0" encoding="utf-8"?>
<pkg build="00" description="Description for test package"
    info="Additional information" name="NameOfThePackage"
    vendor="Vendor" version="00.00.00">
    <scripts src="pkg_scripts/"/>
    <templates src="templates/"/>
    <classes src="pkg_classes/"/>
    <xmls src="pkg_xmls/"/>
    <config path="pkg_config/default.properties"/>
</pkg>
```

1. Here, define the package name, description, vendor, version and build. This may look like:


```
<pkg build="2" description="Test Package"
   info="This is a package built for test purposes" name="MyPackage"    vendor="Company Name"
version="1.0.1">
```

In the above example, the package will be built into the `MyPackage-1.0.1-2.hsp` file.

2. You may not need to edit paths specified in _pkg.xml.

Package Properties File (default.properties)

If you run the package configurator with the `--with-properties` option, package configurator creates the empty `src/pkg_config/default.propeties` file. It has the same syntax as the default H-Sphere properties file `~cpanel/shiva/psoft_config/hsphere.properties`. Package properties will customize the default properties in `hsphere.properties`.

1) You must not change settings in `hsphere.properties`;
2) The package's `default.properties` file should contain only settings to be added to/override the settings in `hsphere.properties`. **Don't copy the whole content of hsphere.properties!**

Templates

The **`src/templates/`** directory with custom templates must have a structure similar to that of the *H-Sphere custom templates directory*.

Important: For those templates that have both `.html` and `.html.in` files, you should include only `.html.in` templates sources into the package. They should be automatically compiled during installation.

Read a separate document on customizing templates with packages.

XML Configuration Files

The **`pkg_xmls/`** directory will include custom XML configuration files, such as `menu.xml` (see *Menu Customization* or `design-config.xml` (see *Skin and Icon*

Set Customization).

If you are planning to include XML configuration files into your package, you must run package configurator with the `--with-xmls --with-properties` options. The `--with-properties` option will create the package properties file, where you will set the corresponding properties for custom XML config files to be included to the package.

1. In default.properties, specify just the names of the XML config files to be customized, without paths to them, for example:

   ```
   DESIGN_SCHEME_CONFIG=design_config.xml
   MENU_CONFIG=menu.xml
   ```
2. Make sure that custom XML config files contain only those tags that need to be customized. Package builder will automatically find and merge custom XML configuration with default configuration.

Read a separate document on customizing XML configuration files with packages.

Language Bundles

If you include *language bundles* to the package, you must run package configuration with the `--with-lang-bundle --with-properties` options. The `--with-properties` option is used to create the package properties file.

1) Add custom language bundle files:

Important: Starting with H-Sphere 3.0 RC 1, `menu.properties` and `messages.properties` become deprecated, and all labels are merged into a single `hsphere_lang.properties` for each language!

- Copy your customized language bundle files to the `src/pkg_lang_bundle` directory (see *Understanding Bundles* for reference):

  ```
  hsphere_lang_<language>_<COUNTRY>.properties
  menu_<language>_<COUNTRY>.properties
  messages_<language>_<COUNTRY>.properties
  ```

  For example, language bundles for Portuguese (Brazil) will be:


```
hsphere_lang_pt_BR.properties
menu_pt_BR.properties
messages_pt_BR.properties
```

- If you are not changing the default language files (`hsphere_lang.properties`, `menu.properties`, `messages.properties`), add empty files with these names to the `src/pkg_lang_bundle` directory:

  ```
  cd src/pkg_lang_bundle
  touch hsphere_lang.properties menu.properties messages.properties
  ```

  This is required for correct *bundle compilation*.
- In `default.properties`, add the following lines:

  ```
  TEMPLATE_BUNDLE=packages.PackageName.hsphere_lang
  MENU_BUNDLE=packages.PackageName.menu
  USER_BUNDLE=packages.PackageName.messages
  ```

  The package properties will be merged with the default settings in `hsphere.properties`.

2) Adding a new language to H-Sphere: see Building New Language Packages

Scripts

The **`pkg_scripts/`** directory will have a tree of subdirectories, their level reflecting the server type, OS family, OS name. For example, if you need to integrate a script that will run on the CP server with RedHat Linux v. 7.3, you will have to put your script into the directory `src/pkg_scripts/CP/Linux/RH7.3/`

Classes

The **`pkg_classes/`** directory will have a tree of Java classes similar to the H-Sphere resource tree in `/hsphere/local/home/cpanel/shiva`.

3. Building

Run package builder to build the package. Use the `--help` option to see the list of possible parameters:

```
java psoft.hsp.tools.PkgBuilder --help
```

The command will look similar to the following:

```
java psoft.hsp.tools.PkgBuilder --with-source=./MyPackage
```

Package builder checks if all required files in the source directory are present, forms one jar file of Java classes (if they are present), and makes a package as one jar file with the `.hsp` extension (e.g., `MyPackage-1.0.1-2.hsp`) in the source directory (`./MyPackage` in the example above).

After the package is built successfully, the package is ready to use. It can be easily *installed* and *uninstalled* on other H-Sphere's.

Related Docs: · Template Customization With Packages · XML Customization With Packages · Building Language Packages

Package XML Configuration File (_pkg.xml)

Related Docs: · Custom Packages · Package Tools

DTD Scheme | Example

Tags and attributes:

- `pkg` - package information. It is set manually in _pkg.xml after package configurator run.
  - ♦ `name` - package name.
  - ♦ `description` - brief description.
  - ♦ `info` - additional info.
  - ♦ `vendor` - package vendor.
  - ♦ `version` - package version.
  - ♦ `build` - package build.
- `scripts` - system scripts to be included into the package. This tag appears if package configurator runs with the `--with-scripts` option.
  Attributes:
  - ♦ `src` - directory in the package `src/` directory where scripts are collected. It is `pkg_scripts` by default. You may not need to change it.
- `classes` - classes to be included into the package. This tag appears if package configurator runs with the `--with-classes` option.
  Attributes:
  - ♦ `src` - directory in the package `src/` directory where classes are collected. It is `pkg_classes` by default. You may not need to change it.
- `jars` - jar files to be included into the package. This tag appears if package configurator runs with the `--with-jars` option.
  Attributes:
  - ♦ `src` - directory in the package `src/` directory where classes are collected. It is `pkg_jars` by default. You may not need to change it.
- `templates` - custom templates to be included into the package. This tag appears if package configurator runs with the `--with-templates` option.
  Attributes:
  - ♦ `src` - directory in the package `src/` directory where templates are collected. It is `templates` by default. You may not need to change it.
- `xmls` - custom XML configuration files to be included into the package. This tag appears if package configurator runs with the `--with-xmls` and `--with-properties` options.
  Attributes:
  - ♦ `src` - directory in the package `src/` directory where custom XMLs are collected. It is `pkg_xmls` by default. You may not need to change it.
- `images` - custom images to be included into the package. This tag appears if package configurator runs with the `--with-images` and `--with-properties` option.
  Attributes:

    - ♦ src - directory in the package src/ directory where images are collected. It is pkg_images by default. You may not need to change it.
- lang_bundles - custom language bundles to be included into the package. This tag appears if package configurator runs with the --with-lang-bundle and --with-properties options.
  Attributes:
    - ♦ src - directory in the package src directory where language bundles are collected. It is pkg_lang_bundle by default. You may not need to change it.
- sql - file with SQL queries to make changes in the H-Sphere database. This tag appears if package configurator runs with the --with-sql option.
  Attributes:
    - ♦ src - file location in the package src/ directory where scripts are collected. It is _SCRIPTS/_pkg.sql by default. You may not need to change it.
- config - custom language bundles to be included into the package. This tag appears if package configurator runs with the --with-properties option.
  Attributes:
    - ♦ path - file pathname relative to the package src/ directory. It is pkg_config/default.properties by default. You may not need to change it.
- actions - container tag for the scripts run before and after the package *installation* and *uninstallation*. They are created in the src/_SCRIPTS directory by default. You may not need to change their location.
    - ♦ pre_install - tag with location of the script that runs before the package installation. This file (src/_SCRIPTS/_pre-install by default) is created empty when package configurator runs with the --with-preinstall option.
    - ♦ post_install - tag with location of the script that runs after the package installation. This file (src/_SCRIPTS/_post-install by default) is created empty when package configurator runs with the --with-postinstall option.
    - ♦ pre_uninstall - tag with location of the script that runs before package uninstallation. This file (src/_SCRIPTS/_pre-uninstall by default) is created empty when package configurator runs with the --with-preuninstall option.
    - ♦ post_uninstall - tag with location of the script that runs after the package uninstallation. This file (src/_SCRIPTS/_post_uninstall by default) is created empty when package configurator runs with the --with-postuninstall option.
- rpms - container tag for the list of rpm tags with third-party RPMs (.rpm files) to be included into the package. The rpms and rpm tags appear when package configurator run with the --with-rpms option.
- rpm - third party RPM file information. During the *package installation* these RPMs will be copied to the /hsphere/shared/scripts/pkg_rpms/ directory to all servers belonging to a logical server group specified in the server_group attribute.
  Attributes:
    - ♦ name - RPM filename without .rpm extension.
    - ♦ server_group - H-Sphere logical server group this RPM relates to: WEB, MAIL, DNSANY applies tarballs to all logical server groups.
- tarballs - container tag for the list of tarball tags with third-party tarballs (.tgz files) to be included into the package. The tarballs and tarball tags appear when package configurator run with the --with-tarballs option.
- tarball - third party .tgz package file information. During the *package installation* these tarballs will be copied to the /hsphere/shared/scripts/3rd_party/ directory to all servers belonging to a logical server group specified in the server_group attribute.
  Attributes:
    - ♦ name - tarball filename without .tgz extension.

- ♦ `server_group` - H-Sphere logical server group this tarball relates to: `WEB`, `MAIL`, `DNS`; `ANY` applies tarballs to all logical server groups.

- `platform` - tags (at least one; may be more than one) included into the `tarball` and `rpm` tags to specify location of third-party tarballs or RPMs for corresponding OS types.
  Attributes:
  - ♦ `name` - server OS type supported in H-Sphere: `RH72`, `RH73`, `RHES`, `RHAS`, `RHAS3`, `FBSD4`, `ANY` for all of the previously mentioned supported OSs.
  - ♦ `location` - tarball/RPM location for this type of platform. Location may be an URL: `http://...` or `ftp://...`, or it may be `BUILT-IN`.
    If `location="BUILT-IN"`, the file will be taken from the OS type subdirectory (the `name` attribute of the `platform` tag) of the tarballs/RPMs directory (`pkg_tarballs` for tarballs or `pkg_rpms` for RPMs in the package's `src/` directory).
    For example: `src/pkg_rpms/RH73/`rpm_name`.rpm` for an RPM file for RedHat 7.3 platform, where rpm_name is set in the `name` attribute of the `rpm` tag.

See also Building Packages guide for package configuration details.

Related Docs: · Custom Packages · Package Tools

Java Tools For Packaging

Related Docs: · Building Packages

Packages are configured, built and installed by means of Java classes located in the directory `/hsphere/local/home/cpanel/shiva/psoft/hsp/tools/`:

- Package configurator: `psoft.hsp.tools.PkgConfigurator`
- Package builder: `psoft.hsp.tools.PkgBuilder`
- Package installer: `psoft.hsp.tools.PkgInstaller`
- Package unistaller: `psoft.hsp.tools.PkgUnInstaller`
- Package checker: `psoft.hsp.tools.PkgRecheck`

Package Configurator

Package Configurator (`PkgConfigurator`) is a tool to configure directory structure for assembling H-Sphere packages. PkgConfigurator creates the package directory skeleton and populates it with files used by the package. The parameters specify the types of the files and where they must be copied from. The source locations must have the correct directory structure. If the path parameter is omitted, the directory skeleton is created empty, and the files will have to be copied manually.

```
SYNOPSIS:
java psoft.hsp.tools.PkgConfigurator
     --with-prefix[=/path/to/directory]
        [   --with-templates[=/path/to/directory]   ]
        [   --with-images[=/path/to/directory]   ]
        [   --with-properties[=/path/to/directory]   ]
        [  --with-xmls[=/path/to/directory]  ]
        [   --with-classes[=/path/to/directory]   ]
        [   --with-scripts[=/path/to/directory] | --with-scripts-advanced[=/path/to/directory]   ]
        [   --with-scripts-advanced   ]
        [   --with-lang-bundle[=/path/to/directory]   ]
        [   --with-preinstall[=/path/to/file]    ]
        [   --with-postinstall[=/path/to/file]   ]
        [   --with-preupgrade[=/path/to/file]    ]
        [   --with-postupgrade[=/path/to/file]   ]
        [   --with-preuninstall[=/path/to/file]   ]
```



```
[   --with-postuninstall[=/path/to/file]   ]
[   --with-jars[=/path/to/directory]   ]
[   --with-rpms   ]
[   --with-tarballs   ]
[   --with-sql[=/path/to/file]   ]
[   --with-sql-uninstall[=/path/to/file]   ]
[   --with-sql-upgrade[=/path/to/file]   ]
[   --with-uninstall-sql[=/path/to/file]  ]
[ - l log_file [ -dl ] ]
```

Allowed parameters:

- `--with-prefix[=/path/to/directory]`

  Specifies the directory where the package skeleton will be generated. If the parameter is omitted, the current working directory will be used as the destination folder.
- `[ --with-templates[=/path/to/directory] ]`

  Specifies that the package contains templates. The template files must be located relative to the custom templates directory. If the path parameter is omitted, the default directory skeleton will be generated in the package's `templates` directory, otherwise, templates located in the specified path will be copied to the package directory.
- `[ --with-images[=/path/to/directory] ]`

  Specifies that the package contains images. The path to the custom directory should be specified.
- `[ --with-properties[=/path/to/directory] ]`

  Specifies that the package requires custom .properties file to be appended to hsphere.properties. If the parameter path is present, the .properties file will be copied from the path specified.
- `[ --with-xmls[=/path/to/directory] ]`

  Specifies that the package uses custom XML configuration files. This allows to customize menu, hosting plans, CP design.
  Important: Run this option with the `--with-properties` option to add custom XML files location into the package properties file.
- `[ --with-classes[=/path/to/directory] ]`

  Specifies that the package contains java classes. This allows to add new classes and override H-Sphere core classes.
- `[ --with-scripts[=/path/to/directory] | --with-scripts-advanced[=/path/to/directory] ]`

Specifies that the package contains system scripts. This allows to add new scripts and override core H-Sphere scripts.

- `[ --with-lang-bundle[=/path/to/directory] ]`

Specifies that the package contains language bundle files. This allows to define new languages and override the default language dependent elements.
Important: Run this option with the `--with-properties` option to add custom language bundle configuration into the package properties file.

- `[ --with-preinstall[=/path/to/file] ]`

Specifies that the package contains a pre-installed script which will be executed before the installation of the package.

- `[ --with-postinstall[=/path/to/file] ]`

Specifies that the package contains a postinstall script which will be executed after the deployment of package files.

- (HS 2.4.3+) `[ --with-preupgrade[=/path/to/file] ]`

Specifies that the package contains the script which will be executed before the the package upgrade.

- (HS 2.4.3+) `[ --with-postinstall[=/path/to/file] ]`

Specifies that the package contains the script which will be executed after the package upgrade.

- `[ --with-preuninstall[=/path/to/file] ]`

Specifies that the package contains a pre-uninstall script which will be executed before the deletion of package files.

- `[ --with-postuninstall[=/path/to/file] ]`

Specifies that the package contains a post-uninstall script which will be executed after the deletion of package files.

- `[ --with-jars[=/path/to/directory]`

Specifies that the package contains Java libraries.

- `[ --with-rpms ]`

Specifies that the package contains third party products supplied as RPMs.

- `[ --with-tarballs ]`

Specifies that the package contains third party products supplied as tarballs (.tgz files).

- `[ --with-sql[=/path/to/file] ]`

Specifies that the package contains SQL file(s) which will be executed against H-Sphere database after the deployment of the package files and before the execution of the package preinstall script if it is pre-configured.

- (HS 2.4.3+) `[ --with-sql-upgrade[=/path/to/file] ]`

  Specifies that the package will have the file with SQL queries to be run upon the package upgrade.
- (HS 2.4.3+) `[ --with-uninstall-sql[=/path/to/file] ]`

  Specifies that the package will have the file with SQL queries to be run upon the package uninstallation.
- `[ -l log_file [ -dl ] ]`

  Creates and writes configuration process messages to a log file; `-dl` is a detailed log mode.

Example:

```
java psoft.hsp.tools.PkgConfigurator --with-prefix=./SuperPKG --with-templates --with-classes --with-scripts
--with-properties -l conf.log -dl
```

The package source files consisting of templates, Java classes, system scripts and the custom properties file will be copied to the `SuperPKG` subdirectory of the current directory.

Package Builder

Package builder is a tool to build a package as one jar file after the package is pre-configured by package configurator and all required tuning is performed. Package builder checks if all required files in the package source directory are present, checks the config file and the package properties file if they are present, forms one jar file from Java classes if they are present and makes a package as one jar file with `.hsp` extension.

```
SYNOPSIS:
java psoft.hsp.tools.PkgBuilder --with-source=/path/to/package/source
```

Example:

```
java psoft.hsp.tools.PkgBuilder --with-source=./SuperPKG
```


Package Installer

```
SYNOPSIS:
java psoft.hsp.tools.PkgInstaller
     --package=/path/to/package/file
     [--upgrade] [--check-only] [--force]
```

Options:

- `--package=/path/to/package/file`

  Specifies the path to the built package. H-Sphere package is a Java archive file (JAR) with `.hsp` extension.
- (HS 2.4.3+) `[--upgrade]`

  Use this option if you upgrade the package already installed on your H-Sphere. With this option, you can easily *upgrade the package* without uninstallation of the older package and restarting H-Sphere after the uninstallation.
- `[--check-only]`

  Makes utility not install the package but only perform a check routine if the package can be installed.
- `[--force]`

  Forces the package installation even if conflicts were detected. For example, if you are installing two packages that use exactly the same file, install the first one without, and the second one with the --force option. The first instance of the file will be replaced with the second. Use this option only if you are sure this won't damage other packages.

See *Package Installation* for details.

Package Uninstaller

```
SYNOPSIS:
java psoft.hsp.tools.PkgUnInstaller
     --pkg-name=PACKAGE_NAME
     [--force]
```


Options:

- `--pkg-name=`PACKAGE_NAME - specify the package name. It should be without the version and build number, as in the Package Name column in *the list of installed packages* (Settings/Packages menu in admin panel).
  For example, if the package installed is `MyPackage-1.0.1-2.hsp`, package name to be specified is MyPackage:

  ```
  java psoft.hsp.tools.PkgUnInstaller --pkg-name=MyPackage
  ```
- `--force` - force the uninstallation.

Package Checker

Starting with 2.4.3 RC 2, scripts, tarballs, and rpms included into an H-Sphere packages are copied to the ~cpanel/shiva/packages/PACKAGE_NAME directory on the CP server. If needed, use java psoft.hsp.tools.PkgRecheck to check package integrity and reinstall the missing files from the *.hsp file.

This is true only for new H-Sphere packages. You will need to copy *.hsp packages already installed on your boxes to the corresponding `~cpanel/shiva/packages/PACKAGE_NAME` directories on the CP server.

```
SYNOPSIS:
java psoft.hsp.tools.PkgRecheck [--pkg-name=PACKAGE_NAME] [--force]
```

Options:

- `[--pkg-name=PACKAGE_NAME]`

  Specifies the path to the package checked; otherwise, checks all installed packages.
- `[--force]`

  Reinstalls missing package files on remote physical boxes.

Related Docs: · Building Packages

Template Customization With Packages

Related Docs: · Building Packages

This documentation dwells on preparation and peculiarities of building the H-Sphere packages containing *templates*.

Preparation

Before you build a package out of your custom templates, make sure customization you need to build into a package has already been performed correctly, that custom directories are correctly defined in the H-Sphere properties file and that custom templates have correct pathes in the custom template directory.

Note: H-Sphere upgrade script would automatically create the custom template directory and the custom image directory if they haven't been created before. If so, just skip certain steps below.

1. Login to the CP server as *cpanel user*.
   All customization should be done under cpanel, and all affected files should have cpanel:cpanel ownership.
2. In the `~cpanel/shiva/psoft_congig/hsphere.properties` file, add or comment out the corresponding lines to specify the correct paths to the directories where you wish to have your custom templates and custom images. Create these directories if they do not exist yet. The default paths are:
   `USER_TEMPLATE_PATH=/hsphere/local/home/cpanel/shiva/custom/templates`
   `USER_IMAGE_PATH=/hsphere/local/home/cpanel/shiva/custom/images`
3. Add symlinks to the custom image directory from the default template directory (`/hsphere/local/home/cpanel/shiva/shiva-templates/`):
   `ln -s /hsphere/local/home/cpanel/shiva/custom/images /hsphere/local/home/cpanel/shiva/shiva-templates/CUSTOM_IMAGES`
   This will create the `CUSTOM_IMAGES` symlink directory in the default template directory.
4. Make sure your custom templates are located in the custom template directory, and your custom images in the custom image directory. To correctly implement new templates or to override default templates or images, the directory structure within the custom directories should be identical to the default directories.
5. To implement custom templates, follow instructions outlined in the *Template Customization* guide.
6. To implement custom images for H-Sphere *skins and icon sets*, assign the `dir` attribute in the `design_config.xml` file to `/CUSTOM_IMAGES` instead of default `/IMAGES`. For example:


```
<icon_image_sets base_dir="/CUSTOM_IMAGES">
```
For details of customizing designs, please read *Skin and Icon Set Customization* in Customization Guide.

Building Packages Containing Templates

After you have prepared custom templates for creating a package, follow the general instructions on building packages, with certain considerations specific to the templates:

- You should include only `.html.in` template sources. They will be compiled automatically during the package installation.
- If you customize the Control Panel menu or skins, you need to run the package configurator with the `--with-xmls` option to add custom `menu.xml` and `design-config.xml` files to the package, and with the `--with-properties` option to plug in these custom XMLs to the system.
  See also Building Packages with XMLs for details.

Related Docs: · Building Packages

XML Customization With Packages

Related Docs: · Building Packages · Template Customization With Packages

This document dwells on building H-Sphere packages to customize XML configuration files.

The following H-Sphere components are configured by means of XML files:

Component	**Property** (in *hsphere.properties*)	**Default Location**
CP Skins (Designs)	DESIGN_SCHEME_CONFIG	/hsphere/local/home/cpanel/shiva/psoft/hsphere/design_config.xml
CP Menu	MENU_CONFIG	/hsphere/local/home/cpanel/shiva/psoft/hsphere/menu.xml
CP Crons	CRON_CONFIG	/hsphere/local/home/cpanel/shiva/psoft/hsphere/cron_config.xml
Online (Context) Help	HELP_CONFIG, ONLINE_HELP_CONFIG	/hsphere/local/home/cpanel/shiva/psoft/hsphere/help.xml /hsphere/local/home/cpanel/shiva/psoft/hsphere/help_url.xml
Promotion Validators And Calculators	PROMO_CONFIG	/hsphere/local/home/cpanel/shiva/psoft/hsphere/promotion/xml/promotions.xml
Plan Wizards	PLAN_WIZARDS_DIR PLAN_WIZARDS	/hsphere/local/home/cpanel/shiva/psoft/plan/wizard/xml /hsphere/local/home/cpanel/shiva/psoft/plan/wizard/xml/plan_wizards.xml
Merchant Gateways	MERCHANT_GATEWAYS_CONF	/hsphere/local/home/cpanel/shiva/psoft/hsphere/merchants.xml
Domain Registrars	REGISTRAR_CONF	/hsphere/local/home/cpanel/shiva/psoft/hsphere/registrar.xml

E-Mail Notifications	`USER_EMAILS`	`/hsphere/local/home/cpanel/shiva/psoft/hsphere/user_emails.xml`

Note:

Some custom XML files can be *merged with default XMLs* by means of XML merger. Instead of moving and changing a default XML file, a small custom file is created containing necessary changes and its location is specified in hsphere.properties in the parameter with the "CUSTOM_" prefix added to the default parameter name, e.g.:

```
MENU_CONFIG = /hsphere/local/home/cpanel/shiva/psoft/hsphere/menu.xml
CUSTOM_MENU_CONFIG = /hsphere/local/home/cpanel/shiva/custom/xml/menu.xml
```

Building Packages With Custom XMLs

1. Log into CP server as *cpanel user*

1. Create Custom XML files

Custom XML files should contain only parts to be added to or to modify default XMLs. During the *package installation* these custom files will be merged with default XMLs. Please refer to the XML Manager guide for examples.

2. Run Package Configurator

Run package configurator in the package directory you created with the following options, at least:

- `--with-prefix=path/to/package/directory`: specify a target directory (relative to the current directory) where all package files will be collected and the package will be configured. We will call it package directory later in the text.
  Hint: You may create a special directory with corresponding package directories where you assemble package source files. For example, it may be `~cpanel/shiva/custom/packages`.
- `--with-xmls[=path/to/xml/directory]`: specify directory with custom XML files. Note that if you don't specify the directory, the empty `src/pkg_xmls` directory will be created in the package directory and you will need to copy custom XML files there manually.
- `--with-properties[=path/to/package/properties/directory]`: this option is required to include custom XML files to the package properties in order to merge custom XMLs with default XMLs on package installation. If you don't specify the directory, the empty

`src/pkg_config/default.properties` file will be created in the package directory.

Thus, package configurator's command line will look like:

```
$ su -l cpanel
bash-2.05a$ java psoft.hsp.tools.PkgConfigurator
--with-prefix=/hsphere/local/home/cpanel/shiva/custom/packages/MyPackage --with-properties --with-xmls
```

3. Configure the Package

To include custom XMLs into the package configuration, add corresponding parameters into the package properties file:

1. Go to the `src/pkg_config` directory of the package directory (`hsphere/local/home/cpanel/custom/packages/MyPackage` in the above example).
2. Edit the `default.properties` file. It will be empty if you did not specify the path to the package properties file in the `--with-properties` option of package configurator.
   - You don't need to copy all contents of the default hsphere.properties file to your package properties file. Only add custom parameters that will be merged with the default ones when the package is installed.
   - In the package properties file, specify just the names of the XML config files to be customized, without paths to the. For example, to include changes in menu and designs, just add these lines to the properties file:
     ```
     # Custom design
     DESIGN_SCHEME_CONFIG=design_config.xml
     # Custom menu
     MENU_CONFIG=menu.xml
     ```
     Parameters corresponding to each XML configuration file are given in the above table.

After you have configured package properties, proceed to other package configuration options required.

4. Run Package Builder

Run package builder to build the package, for example:

```
java psoft.hsp.tools.PkgBuilder --with-source=/hsphere/local/home/cpanel/custom/packages/MyPackage
```

The package will be assembled to the .hsp Jar file in the current directory.

5. Install the Package

Login as cpanel and run *package installer*, for example:

```
java psoft.hsp.tools.PkgInstaller --package=./MyPackage
```

Related Docs: · Building Packages · Template Customization With Packages

Building Language Packages

Related Docs: · Building Packages

This document will guide you through the process of building a package that adds a new language to H-Sphere interface. A package is built into one portable `.hsp` file and can be easily *installed* by an H-Sphere system administrator.

Please also refer to PSoft collection of language packages.

To build a language package, you should be familiar with the concept of *language bundles* and the way they are *compiled*.

Let us take Portuguese (Brazil) as a sample language to be added. We assume you have the language bundles already *translated*.

1. Log into the CP server as *cpanel user*.
2. Create a special location where you will assemble and build your packages, for example, `~cpanel/shiva/custom/Packages`:

   ```
   mkdir ~cpanel/shiva/custom/Packages
   ```
3. Run package configurator:

   ```
   cd ~cpanel/shiva/custom/Packages
   java psoft.hsp.tools.PkgConfigurator --with-prefix=./pt_BR --with-properties --with-lang-bundle
   ```

   Package configurator will create the `pt_BR` directory, which includes the following structure:

   - `src/_pkg.xml` - package configuration file
   - `src/pkg_lang_bundle/` - directory where you will place new language files
   - `src/pkg_config/default.properties` - custom package properties file (initially empty)
4. Set the package name, version, build, vendor, description in `src/_pkg.xml`:

   ```
   <pkg build="1" name="Language_PT_BR" description="Portuguese (Brazil) Language Package"
      info="Package for installation of the Portuguese (Brazil) language into H-Sphere"    vendor="Sample
   Company Inc." version="0.0.1">
   ```

   You don't need to edit other tags in `_pkg.xml`.

5. Copy the translated language files into `pt_BR/src/pkg_lang_bundle/`:

   Important: Starting with H-Sphere 3.0 RC 1, `menu.properties` and `messages.properties` become deprecated, and all labels are merged into a single `hsphere_lang.properties` for each language!

   ```
   cd src/pkg_lang_bundle/
   cp /some/location/hsphere_lang_pt_BR.properties .
   cp /some/location/menu_pt_BR.properties .
   cp /some/location/messages_pt_BR.properties .
   ```
6. Create empty files `hsphere_lang.properties`, `menu.properties`, and `messages.properties` in the `pt_BR/src/pkg_lang_bundle/` directory:

   ```
   touch hsphere_lang.properties menu.properties messages.properties
   ```

   These files are required for correct bundle compilation during the package installation.
7. Add the `misc.langs.<LABEL>lang` label with the name of the new language into `hsphere_lang.properties`. This text will appear as the new language option in the Change Language dropdown menus in H-Sphere.

   ```
   misc.langs.ptlang = Portugu\u00eas (Brasil)
   ```

   Notes:

   1) Default (English) language bundles are written in the ISO-8859-1 encoding. Special Latin and non-Latin characters must be converted into Unicode. Use the native2ascii JDK tool to convert these symbols into Unicode characters (`\uxxxx` notations, like "Portugu**\u00ea**s" instead of "Portugu**ê**s" in the example above).

   2) If the original language name significantly differs from that in English, (especially for non-Latin alphabets), we recommend adding its English transcription, e.g.:

   ```
   misc.langs.de_atlang = Deutch (\u00d6sterreich) - German (Austria)
   ```
8. Edit `src/pkg_config/default.properties`:

   1. Add properties that specify location of language bundles implemented in this package. The format is:

      ```
      TEMPLATE_BUNDLE=packages.PackageName.hsphere_lang
      MENU_BUNDLE=packages.PackageName.menu
      ```



```
USER_BUNDLE=packages.PackageName.messages
```

Here, PackageName is the value of the `name` attribute of the `pkg` tag in `_pkg.xml`.

In our example, we add the following lines to `default.properties`:

```
TEMPLATE_BUNDLE=packages.Language_PT_BR.hsphere_lang
MENU_BUNDLE=packages.Language_PT_BR.menu
USER_BUNDLE=packages.Language_PT_BR.messages
```

2. Add the new language to the list of languages available in H-Sphere. Set the language and encoding of the translated language files in the `LANG_LIST` parameter. The format is:

```
LANG_LIST = <language>_<COUNTRY>_<ENCODING>|<HTML_ENCODING>:misc.langs.<LABEL>lang
```

Here:

- `<language>`, `<COUNTRY>`, and `<ENCODING>` are Java locale identifiers. For detailed description and the tables of canonical identifiers, please refer to *Understanding Language Bundles* in Customization Guide.
- `<HTML_ENCODING>` is the HTML-compliant encoding (this parameter is deprecated since 2.4 and is not really used but must still be specified for the sake of compatibility);
- `misc.langs.<LABEL>lang` is the previously set language label.

For Portuguese (Brazil):

LANG_LIST = pt_BR_ISO-8859-15|ISO-8859-15:misc.langs.ptlang

Notes:

1) Don't specify other languages into `LANG_LIST` in `default.properties`! During the package installation, the package properties will be customize the default properties in `~cpanel/shiva/psoft_config/hsphere.properties`.

2) You must specify the correct encoding in which the language bundle files were created and saved. During the package installation, *language bundle compiler* will convert these files into UTF-8.

9. Return from the package directory and run package builder:

```
cd ~cpanel/shiva/custom/Packages
java psoft.hsp.tools.PkgBuilder --with-source=./pt_BR
```


The package `Language_PT_BR.0.0.1-1.hsp` will be created in the `pt_BR` directory. It is ready to be *installed* on H-Sphere.

Related Docs: · Building Packages

Creating Resources for Winbox

Introduction

In terms of H-Sphere, any resource on a Windows server can be considered as a module (or a group of modules), and has to support the following functionalities:
- create and delete a physical resource;
- read and change parameters of the physical resource.
By physical resource, we mean any resource of the OS (IP address, account, etc), or additional software used as a part of the hosting service. Different physical resources can have the same functionality in terms of hosting (even if they are implemented in completely different ways), like Serv-U FTP server and MS FTP Server. Even for the same software, different versions of the physical resource implementation can exist with different functionality. As a result, each H-Sphere resource can have several implementations, i.e. modules working with a particular implementation of a physical resource. Yet, only one implementation of a particular H-Sphere resource can be active at a time. To put it simply, we will be using a resource to describe its physical implementation.

Requirements

To implement a new resource, the following is required:
- .NET Framework version 1.0.3705 and higher
- H-Sphere Windows package Installed or Psoft.HSphere.dll assembly available
You can use any .NET compatible language. Yet, since the core part of H-Sphere winbox is written in C#, all examples will be presented in C#.

Naming conventions

We recommend using the following naming conventions when developing new resources.

Namespaces

All resources should belong to Psoft.Hsphere.Resources or to namespaces inside it. If there are several implementations of the same resource, they should be placed in the same namespace.

Assemblies

The name of the assembly should have the following format:
Resources.<Resource class name> if assembly contains class of only one resource and the resource is located directly in the namespace Psoft.HSphere.Resources (e.g.: Resources.Diskquota for resource Psoft.HSphere.Resources.DiskQuota) Resources.<Namespace Name> if assembly contains several resources in this namespace. (e.g.: Resource.IIS.ColdFusion for the implementation of different versions of the ColdFusion resource)

Attributes

All names in attributes (logical name of the resource, physical name, parameter names in schemas) should be typed in lower case.

General Scheme of work for the resource

Each resource has a logical name known to the CP. If there are several implementations of the same resource, only one can be active at one time. The implementation is defined in the config file. The implementation of the resources should be in separate non-core assemblies that will be loaded dynamically. The class that implements resource and its assembly is defined in the config file. Also, in the config file, the attributes of the resource are defined, and each instance of the resource will get those attributes in addition to parameters. The attributes should represent constant parameters for a given implementation that are common for all instanaces. The resource (group of resources) has to be implemented as a public class (classes), inherited from abstract class Psoft.HSphere.Resources.Resource that can be found in the Psoft.HSphere.dll assembly and implemented as class library assembly. To do that, you need to add references to Psoft.HSphere.dll assembly during the compilation using option/reference.

Class Psoft.HSphere.Resources.Resource defines the set of methods that have to be implemented by the inherited resource:

```
public abstract void Create();
public abstract void Get();
public abstract void Update(Parameters newParams);
public abstract void Delete();
public abstract void Suspend();
public abstract void Resume();
```

The names of the methods describe their functionality (more details below). Besides that, each resource inherits a collection of parameteres from the abstract parent of the Psoft.HSphere.Transport.Protocols.Parameters type. That collection is expressed as a hash table, and the values of parameters can be retrieved via the parameter names. When a particular operation is executed on the resource (like create, delete, update parameters... etc), H-Sphere Windows engine creates instance of the corresponding resource and sets values for the parameters as they were passed from the CP. Besides that, the resource can have a collection of configurable attributes,

which are defined in H-Sphere config file. H-Sphere Windows engine adds the attributes into collection of parameters as well. After that, the corresponding method is called on the resource. Due to that, during the call of any of those methods, instance of the resource has all the required information to perform the operation.

During the development of the resource, you have to consider that resource does not support statuses, and you should not save any type of info in the instance of the resource, as it can be destroyed as soon as the method is executed.

Methods

The above methods have to be implemented, as they are defined as abstract.

void Create()

This code should implement physical creation of the resource. Parameteres required to create the resource will be passed as parameters collection. The number and names of parameters are defined by specifics of physical resource and set in special attributes-schemas (defined later in this document).

void Get()

This method should implement the code to retrieve parameters of a physical resource. Retrieved values should be written into collection parameters. If such functionality is not required for the resource, it can be implemented as empty method:

```
void Get() {}
```

void Update(Parameters newParams)

This method should implement a way to change all or some parameters of a physical resource. The method accepts a collection of parameters that need to be changed. During the implementation of this method, you don't have to update the collection resource parameters (it will be done by H-Sphere Windows engine if the code is completed successfully). It is enough to update parameters of the physical resource.

If the given functionality is not required for the resource, it can be implemented as an empty method:

```
void Update(Parameters newParams) {}
```


void Delete()

This method should implement the deletion of the physical resource. If the given functionality is not required for the resource, it can be implemented as an empty method:

```
void Delete() {}
```

void Suspend()

This method should implement the suspension of the physical resource. If the given functionality is not required for the resource, it can be implemented as an empty method:

```
void Suspend() {}
```

void Resume()

This method should resume the physical resource. If the given functionality is not required for the resource, it can be implemented as an empty method:

```
void Resume() {}
```

Keys

H-Sphere Windows engine and resource class deal with terms like "resource keys". There are two types of keys, logical and physical. Logical key can be defined as a parameter which differentiates instance of the resource from the set of other instances for the resource user (i.e. CP). To operate with particular instance of the resource, in addition to all other parameters added to its methods, you need to pass the parameter that will uniquely identify the resource. Physical key is very much like a logical key, with the only difference that resource user (i.e. CP) knows nothing about it (there is no unique identifier). Sometimes physical resources require keys to operate with them, which makes sense only in the context of the given physical key. It can be exemplified by numeric IDs of virtual web or ftp hosts (unique numerical identifiers) in the IIS metabase, that are required to operate with a given host on the level of metabase.

On the other hand, for virtual web host uniqueness can be implemented by means of domain name, and for virtual FTP host - by means of IP. In that case domain name is a logical key for the resource that implements virtual web host. A number in the metabase - its resource's physical key. At the same moment H-Sphere Windows engine provides mechanisms for the translation of virtual key into physical key. In most cases, logical key can coincide with physical key.

As mentioned above, a resource has two types of parameters that eventually are stored in common collection of parameters - configuration (common to all instances) and instance (different for each particular instance of the resource). Instance parameters are passed during the method call and should contain logical key for the given resource.

Each key is implemented via subclass of abstract class Psoft.HSphere.Resources.Key. This class defines the following set of methods to be overloaded:

```
public abstract object Val { get; set; }
public abstract int Compare(object objVal);
public abstract Key Add(object objVal);
public abstract Key Sub(object objVal);
public abstract Key Inc();
public abstract Key Dec();
```

Those methods define basic operations for manipulating and comparing keys. In most cases, you don't need to create a new key class, as you can use two already existing classes: Psoft.HSphere.Resources.NumKey and Psoft.HSphere.Resources.StrKey, that correspondingly implement numeric and string keys.

Class Resource has two attributes, Pkey and Lkey, to represent physical and logical keys for the instance of the resource:

```
public Key PKey { get; set; }
public Key LKey { get; }
```

LKey is read-only, as the value of the logical key is automatically retrieved from the parameters collection.

Error handling

All the classes implementing exceptions that are connected with resources have to be inherited from class Psoft.HSphere.Resources.ResourceException. There are several types of error conditions that should be indicated when manipulating with resources:
- attempt to create resource that already exists
- attempt to do an operation on the resource that doesn't exist
- error in the data passed as parameters.

The first error type can appear only in method Create(). For that type, there is a class Psoft.HSphere.Resources.ResourceAlreadyExistsException inherited from ResourceException. During the implementation of the Create() method, you have to make sure that given exception is thrown in the situation when the physical resource exists and from the point of view of the resource's logic it is considered as an error. (For some resources, given condition may not be an error, e.g error that appears while adding an IP that has been already set up before can be safely ignored).

The second type of error conditions can appear in any method, but Create(). That type of errors is described by Psoft.HSphere.Resources.ResourceNotFoundException class. During the implementation of such methods, you have to make sure that those conditions are taken care of, and that the exception is thrown.

The third type of conditions is described by class Psoft.HSphere.Resources.ResourceInvalidParameterException. That type is used to describe situations when the problem is created by incorrect data entered by end user - to allow end user to fix them.

Class Resource has got a set of methods to generate exceptions of these types:

```
protected void ErrorNotFound()
protected void ErrorNotFound(string message)
protected void ErrorAlreadyExists()
protected void ErrorAlreadyExists(string message)
protected void ErrorInvalidParameter(string pName)
```

Resource Hierarchy

Based on H-Sphere logic, each resource belongs to hierarchy of resources. This hierarchy shows dependence of one resource on another. For example, to create the virtual webhost resource, it is required to specify an account name that will be the owner of that virtual host. It means that virtual host resource depends on resource account, and stands below it in the hierarchy of resources.

During the development of resource class, you can specify the place of the resource in the hierarchy using special attribute (attributes will be explained in details below). Even though the configuration of resource attributes is located in the config file, each resource has access to only its own set of attributes. Yet, it also inherits configuration parameters of its parent resource (in the hierarchy). Therefore, resource gets configuration parameters of all its parents in parameters collection.

This resource's quality does not depend on the instance of the resource, such as configuration parameters are common for all instances of the resource.

There is another aspect of using resource hierarchy. The child resource can access parent resource to retrieve parent resource's instance parameters. Each resource has instance parameter which is its logical key. That parameter uniquely identifies the instance of this resource. We can connect child resource with parent resource by means of parameter from the instance parameters set, that has the same name as key parameter. In such child resource we have access to instance of parent resource, using the Parent attribute inherited from Resource:

```
public Resource Parent { get; }
```

Accordingly, by means of the Pkey attribute, physical key of the parent resource instance can access the collection of parameters via the Params' property:

```
public Parameters Params {get; set;}
```


Logging

Each resource inherits method Log that can be used for logging. The log output will be placed in the common log file resource.log. Method has the following signature:

```
protected void Log(string format, params object[] args)
```

Resource Types

Resources can be of two types: those that support listing and those that don't support listing. In the case of listable resource, H-Sphere Windows engine will load all instances of the resource into cache. Cache will be used to consequently retrieve instances of the given resource. Listable resources do two important things: the information is cached that allows to optimize method Get(), and such resources support automatic generation of new physical keys during creation of resource. If resource has to support physical keys and it requires their automatic management, resource has to be listable. To notify that resource is listable, you have to overload a static method:

```
ArrayList Enum(Key parPKey, Parameters configParams, ResourceLogger log)
```

This method has to return list of physical keys for existing instances of resource. As parameters, the value of physical key of corresponding parent resource is passed (if there is no parent resource, or if parent resource doesn't support physical keys, the parameter can be ignored), configuration resource parameters and log object.

Resources that don't support physical keys can be listable as well.

Method Enum has to return list of logical keys, as they are also physical keys for those resources.

Listable resources support automatic access synchronization to their instances. They also provide additional level of checking the existing resource during creation, and missing resource check during other operations using cache. They will automatically generate ResourceAlreadyExistsException or ResourceNotFoundException exceptions.

Schemas

Schemas are used not only in resources, but we will discuss them only in the context of Resource. Schemas are descriptions of attributes of a particular object. Schemas are implemented by class Psoft.HSPhere.Configuraiton.Schema. Constructor of the given class accepts XML string as a parameter. That XML string describes the given

schema. XML schemas consist of the following elements: schema tag, properties tag (prop), attributes of properties. Schema tag contains a prop tag which can have attribute tags. The prop tag has the following attributes:

- "name" - required attribute, it defines name of the property
- "description" - contains short description of the property
- "type" - contains identifier of the property type. The following identifiers are supported:
    - str - string (default)
    - num - numeric
    - unum - non-signed numeric
    - bool - boolean
    - select - list (will be described bellow)
    - path - file system path
    - password - password
    - ip - IP address
    - domain - domain name
    - url - URL
    - mail - mailbox name
    - mask - IP mask
- mac - MAC address
- value - default value of the property
- key - false by default. If false, it is not the key property. If true, property is a key property.
- Optional - false by default. If false the property is optional, otherwise it is required.
- "readonly" - deafult false. If false, the property can be changed (can be passed as the parameter), if true, the property is read-only.

If parameter has type "select", this tag has to contain list of tags that describe values of the list. Values by default are set to be equal to the name of the value.

There are two types of schemas for resources - schemas that describe configuration parameters, and schemas that describe instance parameters. A resource doesn't have to have a configuration parameters schema (for example, if it doesn't have config parameters), but schema of instantace parameters must always be defined.

Based on that schema, the input is validated on the presence of key and required parameters. The default values are also set at that stage for missing parameters. If the parameter is described in both config and instance schemas, then if parameter wasn't passed (it is possible, if it was described as optional), the value for it will be taken from the config file, otherwise, the input value has higher priority and is used as the parameter. Examples of schemas are in the section "Attributes".

Attributes

Resource has a list of attributes, some of them are required. Attributes define properties of the resource and define its behavior.

ResourceName

Required attribute that specifies logical name of the resource. It will be passed as the name to invoke that resource by the CP.
Example: `[ResourceName("hosting")]`

PhysicalName

Required attribute that contains implementation name of the given logical resource.
Example: `[PhysicalName("iis5_website")]`

LKey

Required attribute that defines logical key for the resource, or, more specifically, the name of the key parameter of the implementing class.
Example: `[LKey("hostname", typeof(StrKey))]`

ResourceDescription

Optional attribute that contains short description of the resource.
Example: `[ResourceDescription("IIS 5.0 virtual web host")]`
Even though this attribute is not required, we highly recommend using it.

ParentName attribute

Optional attribute that contains logical name of the parent resource.
Example: `[ParentName("account")]`
This attribute defines the place of the resource in the resource hierarchy. If it is not defined, the resource is considered to be root in the hierarchy.

ManagedPKey attribute

Optional attribute. Shows if physical keys will be automatically managed.
Example: `[ManagedPKey]`
If the attribute is not defined, the resource has to generate unique physical keys. If the resource doesn't support physical keys, this attribute is meaningless.

ResourceSchema attribute

Optional attribute. Describes the configuration schema and input parameters of the resource. Example:

```
[ResourceSchema(
@"<schema>
        <prop name='hostname' type='domain' description='Virtual web host name'/>
        <prop name='ip' type='ip' description='Virtual web host IP'/>
        <prop name='port' type='unum' value='80' description='Virtual web host port'/>
        <prop name='username' description='Owner account of virtual web host'/>
        <prop name='docroot' optional='true' description='Document root name of virtual web host'/>
        <prop name='docrootpath' readonly='true' description='Document root directory full path'/>
        <prop name='status' type='select' value='starting' description='Status of virtual web host'>
                <prop name='starting' value='1' description='Starting virtual web host'/>
                <prop name='stopping' value='3' description='Stopping virtual web host'/>
                <prop name='pausing' value='5' description='Pausing virtual web host'/>
                <prop name='continuing' value='7' description='Continuing virtual web host'/>
        </prop>
        <prop name='index' type='bool' value='false' description='Indexing flag'/>
        <prop name='idomian' optional='true' type='domain' description='Instant alias of virtual web host'/>
        <prop name='dedicatedip' type='bool' value='false' description='true if IP property is dedicated IP'/>
        <prop name='ss_lid' optional='true' description='Site Studio lid'/>
        <prop name='applevel' type='select' optional='true' description='Web applications isolating level'>
                <prop name='inproc' value='0' description='Inproc application'/>
                <prop name='outproc' value='1' description='Outproc application'/>
                <prop name='pooled' value='2' description='Pooled application'/>
        </prop>
</schema>",
@"<schema>
        <prop name='logsdir' type='path' description='Log files directory'/>
        <prop name='suspendskeleton' type='path' value='\skeleton\suspend' description='Suspned skeleton location'/>
        <prop name='inheritdefaultdocs' type='bool' value='false' description='Direct to inherit default documents

    from default web properties'/>
        <prop name='removecontent' value='false' type='bool' description='Force to remove all content from deleted
```



```
    website'/>
        <prop name='enableparentpath' value='true' type='bool' description='If true using of absolute pathes (such

    as \somedir\somescript.asp) in the ASP scripts will be enabled'/>
        <prop name='defaultdocs' value='default.htm, default.asp, default.html, index.html, index.htm, default.stm,

    index.stm, default.shtml, index.shtml, default.shtm, index.shtm, default.aspx, index.aspx' description='List

    of default documents'/>
        <prop name='applevel' type='select' value='pooled' description='Web applications isolating level'>
                <prop name='inproc' value='0' description='Inproc application'/>
                <prop name='outproc' value='1' description='Outproc application'/>
                <prop name='pooled' value='2' description='Pooled application'/>
        </prop>
</schema>"
 )]
```

This attribute has two constructors. The first constructor accepts both schemas as shown in the example. The second constructor is used for resources that don't have configuration parameters and have only one schema.

Exclusive

Optional attribute. Specifies if synchronization should be used to access different instances of the resource.
Example: `[Exclusive]`
If the attribute is not defined, synchronization will not be performed. If the resource is listable, the attribute is ignored, because listable resources automatically support synchronization.

Offline

Optional attribute. Shows that all the operation on the resource has to be performed offline.
Example: `[Offline]`
Currently not implemented.

IgnoreExists

Optional Attribute. Affects the behaviour when there's an attempt to create an already existing resource.
Example: `[IgnoreExists]`

If the attribute is set, the already existing condition is not considered to be an error.

Cacheable

Optional attribute. Affects the cache behaviour during data access from cache for listable resources.
Example: `[Cacheable(false)]`
The default is true. If set to false, the data will not be caught. This can be useful for resources with read-only parameters, such as "usage" for disk quota. For non-listable parameters, this attribute is ignored.

CaseSensitive

Optional. Defines if logical keys are case sensitive.
Example: `[CaseSensitive]`
False by default.

IIS 6.0 Native Mode

H-Sphere is compatible with IIS 6.0 native mode. H-Sphere automatically switches IIS 6.0 to native mode during the installation or update procedure. After this, you need to restart IIS to apply changes.

In addition, H-Sphere switches the identity of all existing application pools to "Local System" account instead of the default "Network service" account. It doesn't manage application pools at this time, so if a new application pool is created, its identity should be manually set to "Local System" account.

There are several ISAPI filters and IIS log plugins that are part of H-Sphere which are implemented in the modules described below.

ISAPI Filters

Since IIS 6.0 in native mode has quite different architecture regarding the ISAPI filters and log plugins, these modules were rewritten to accomplish compatibility with IIS 6.0 native mode simultaneously with IIS 5.0 support.

- **`HsAuth.dll`** - ISAPI filter intended to realize several of web authentication schemes such as web authentication for virtual accounts (for Serv-U users), web authentication for FrontPage and cookie authentication.

  H-Sphere uses a user account as an anonymous account for the user domain instead of the standard built-in IUSR account to avoid unapproved content changes. For SERV-U FTP, FrontPage reads the anonymous account settings for the virtual host and treats this account as an account which has the anonymous access to FrontPage. Therefore H-Sphere sets the Basic authentication to the FrontPage virtual directories and installs the HSauth filter to authenticate the user when he connects via FrontPage client.
- **`HtAccess.dll`** - ISAPI filter intended to realize folder protect feature for WebShell 4.0.

  H-Sphere does not support 'ISAPI_Rewrite' filter because it processes the URL and then changes it. But in the current security model of HTProtect work such operation is not allowed . So 'ISAPI_Rewrite' and 'HTAccess' filters conflict with each other when both are running in IIS.
- **`SharedSSL.dll`** - ISAPI filter intended to realize Shared SSL functionality.

  To process https requests, we created service virtual hosts for each IP where shared SSL service is used, including dedicated IPs. A service virtual host home dir contains virtual directories for websites that have shared SSL enabled. Such a virtual directory bears the name of the third level domain alias and points to the document root of the corresponding website. ISAPI shared SSL filter, which is installed on every service virtual host, redirects https requests to the appropriate virtual directory. (Read more about Winbox Shared SSL)

IIS Log Plugin

Now due to changes in H-Sphere log plugins, even currently modified HTTP log files can be opened for writing or removed.

`HsLogPlugin.dll` - module which contains several IIS log plugin intended to realize stats collecting on the fly, different schemes of web and ftp logging such as simple web logging, transfer log, etc.

Crash Reporting on Winbox

Questions Considered:
- understanding Winbox crash reporting
- crash record file structure
- detecting crashes in H-Sphere and non H-Sphere modules

Understanding Winbox Crash Reporting

H-Sphere Winbox writes unhandled exceptions to files iis.log and hssvc.log that are located in the `<H-Sphere dir>\logs\crash` directory. The messages written to these logs aren't very informative and are barely helpful in debugging unless you install the pdb package that complements error logs with detailed debug information. To install the pdb package, see how to update Winbox to the latest stable version. The pdb package can send crash reports to Positive Software support - the installer will prompt you for the SMTP server.

Crash Record File Structure

Each crash record file is of the following structure:
- the 1st line contains a crash date and crash description;
- the rest of the lines contain a crash stack.

Example:

```
[26.07.2004 / 16:36:33] Thread: 27904; Access violation occurred in module inetinfo (image )
   at RtlAllocateHeap() in :line 0 (in module ntdll)
   at () in :line 0 (in module htaccess)
   at () in :line 0 (in module htaccess)
   at () in :line 0 (in module htaccess)
   at ?BuildURLMovedResponse@HTTP_REQ_BASE@@QAEHPAVBUFFER@@PAVSTR@@KH@Z() in :line 0 (in module w3svc)
   at ?BuildURLMovedResponse@HTTP_REQ_BASE@@QAEHPAVBUFFER@@PAVSTR@@KH@Z() in :line 0 (in module w3svc)
   at ?ResetSSLInfo@W3_SERVER_INSTANCE@@SGXPAX@Z() in :line 0 (in module w3svc)
   at ?Disconnect@CLIENT_CONN@@QAEXPAVHTTP_REQ_BASE@@KKHPAH@Z() in :line 0 (in module w3svc)
   at ?ScanForTerminator@@YGPAEPBD@Z() in :line 0 (in module w3svc)
   at ?Copy@STR@@QAEHPBDK@Z() in :line 0 (in module w3svc)
```


at ?RemoveEntry@CDirMonitor@@QAE?AW4LK_RETCODE@@PAVCDirMonitorEntry@@@Z() in :line 0 (in module ISATQ)
at () in :line 0 (in module)

Each crash stack line may contain:

- information about the function being executed when the crash occurred;
- source information, such as a source file's name and line's number
  * Note: to record H-Sphere module's source information the pdb package is required (see how to update Winbox to the latest stable version).
- module name containing crashed function

Detecting Crashes in H-Sphere and non H-Sphere Modules

The name of the module in the crash stack line indicates which module the crash occurred in. In the example above (in module htaccess) indicates that the crash occurred in H-Sphere module. An empty module file's name, for example (in module) indicates that the crash occurred in the main module.

When a crash occurs in any module, the crash report framework generates an exception containing information about the crash. A crash in H-Sphere module will trigger the exception and a record with H-Sphere module name will be written to the report file, for example (in module htaccess). As non H-Sphere modules can't catch the exception generated by the crash report framework, it will return back to crash report as "C++ exception" and will be registered in the crash report file immediately after the initial crash record. In this case two records will be written into the report file: the initial crash record and the "C++ exception" record as shown in the following example:

[26.07.2004 / 16:36:33] Thread: 3432; Access violation occured in module w3wp (image)
at () in :line 0 (in module)

[26.07.2004 / 16:36:33] Thread: 3432; C++ exception occured in module w3wp (image)
at RaiseException() in :line 0 (in module kernel32)
at () in :line 0 (in module htaccess)
at () in :line 0 (in module htaccess)
at UnhandledExceptionFilter() in :line 0 (in module kernel32)
at FlsSetValue() in :line 0 (in module kernel32)

Two records with the same crash date, time and crash description indicate that the crash occurred in a non H-Sphere module.
* Note: the time for the second record can differ by several seconds.

Creating Plan Wizards with XML

This document explains how to customize plan wizards by modifying plan wizard XML configuration files.

Introduction

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 `PLAN_WIZARDS_DIR` in `~cpanel/shiva/psoft_config/hsphere.properties` to a target directory.

PLAN_WIZARDS_DIR = /hsphere/local/home/cpanel/shiva/psoft/hsphere/plan/wizard/xml

The list of wizards is defined in the `plan_wizards.xml` file, which is by default located in the plan wizards directory. The file's name and location is set in the `PLAN_WIZARDS` parameter in hsphere.properties (the full path to the file is required):

PLAN_WIZARDS = /hsphere/local/home/cpanel/shiva/psoft/hsphere/plan/wizard/xml/plan_wizards.xml

IMPORTANT:
When you customize plan wizard XMLs make sure the `PLAN_WIZARDS_DIR` and `PLAN_WIZARDS` parameters are set in `hsphere.properties`.
You can also customize plan wizard XML files by means of H-Sphere packages.

Adding A New Wizard To The List Of Plan Wizards

To add a new wizard, add the following line to the list of wizards in `plan_wizards.xml`:

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

Here,

- `name` - the name of a plan's XML file. It must be specified without `.xml` extension (.xml will be appended automatically). The file contains plan wizard specification.
- `description` is a language bundle label defined in `~cpanel/shiva/psoft/hsphere/lang/hsphere_lang.properties`. It should not contain the "lang." prefix.
  Read more about *bundles* in Customization Guide.

Example:

<wizard name="unix" description="planeditor.res_unix"/>

Here, Unix plan XML definition file is `~cpanel/shiva/psoft/hsphere/xml/unix.xml`, and the plan's description is set in the `lang.planeditor.res_unix` label in `~cpanel/shiva/psoft/hsphere/lang/hsphere_lang.properties`.

Defining Plan Wizard

Plan wizard definition starts with creating a new .xml file. See unix.xml as an example of planwizard definition.

The root XML tag is:

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

Attributes:

- `name` - should match the filename (without .xml prefix) and the corresponding `name` attribute in plan_wizards.xml.
- `description` - language bundle with plan wizard description from hsphere_lang.properties without "lang." prefix.

The `PlanWizard` construction includes the following tags:

`DefaultName` is the name that will serve as the default plan name in creating plans.

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


DefaultValues is a construction where the plan's default values are set. Usually, the add _template and menuId values are added here.

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

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

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

The following tags are used inside the category tag:

<resource>

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

- name - name of the resource, like account, mailbox;
- class - name of the resource class;
- help (optional) - description of the resource from hsphere_lang.properties file;
- unit (optional) - units for the resource, MB or GB;
- required (optional) - if it is set to 1, the resource is required in the plan and regarded to be automatically included to the plan;
- include (optional) - if it is set to 1, the 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.

`<field>`

Inside the `<resource>` construction, the `<field>` tag allows to get more info from the user for a resource specified. 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` - label from hsphere_lang.properties, refers to a 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.

<LogicalGroup>

The <LogicalGroup> tag 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"/>
```

<ifresource>

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.propeperties.

<ifgroup>

The <ifgroup> element works in 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.

`<resources>`

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 the unixuser resource the tag would look like:

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

`<mod>`

The `<mod>` tags determine what mods will be defined in the plan.

Attributes:

- `name` - name of the mod; "" if missing;
- `ifresource` - list of resources delimited with '|'. 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 won't be defined.

Inside the `mod` element the following tags are structured:

`<initresource>`

The `<initresource>` tags inside `mod` constructions define what child resources are created with the creation of this resource.

Attributes:

- `name` - name of the resource. The system automatically checks if the resource is included in the plan before adding the initresource;
- `ifactive` - list of resources delimited with '|'. If any of these resources are selected in the plan wizard as activated, the initresource is created; otherwise, it won't be created, but will be available for creation by account owners.
- `unique` (optional) - a flag to mark the initresource whose mod will be changed in the plan wizard. For example, you may set initresource for the IP resource to shared IP, and then change shared IP to dedicated IP in the plan wizard in Control Panel. In this case, if you don't set the `unique` attribute in the `initresource` tag, another initresource record will be added to the system database and that will cause serious malfunctions.
  However, if `unique="1"` is set, no new initresouce record will be created when another mod is selected; instead, the existing initresource record will be modified.
  Example:

  <initresource name="ip" mod="shared" unique="1"/>

`<initvalue>`

The `<initvalue>` tag defines initial plan values.

Initvalues are passed to the resource constructor as a Collection, and you need to be very careful with their order. Initvalue names aren't used as keys. Typically, the constructor would read initvalues like this:

Iterator i = initValues.iterator();
value1 = (String) i.next();
value2 = (String) i.next();

Attributes:

- `type` - one of the following types as defined in psoft.hsphere.plan.InitValue:
  - static - the value enclosed in the tag.
  - field - the value taken from the submitted form by the name enclosed in the tag.
  - relative - the value obtained with the 'TemplateModel get(String key)' method of the resource chain from the current resource up the resources tree. The search stops upon finding the first not null value.
  - absolute - the value obtained with the 'TemplateModel get(String key)' method of the current resource. Unlike 'relative', it allows geting values of TemplateHash values returned by 'TemplateModel get(String key)'.
    Example: Let's say a resource returns a hash accessible from TemplateModel get(String key) by key 'a'. This hash, in its turn, contains a value accessible by key 'b'. To access this value, you need to set initvalue to 'a.b'.

- relative_rec - implements embedding as in 'absolute' with support of ascending recursion as in 'relative'.
- absolute_rec - implements embedding, but calls 'TemplateModel get(String key)' of the top resource in the tree - the account.
- hostgroup - is used to get a random HostEntry object within the given logical server group. The value of the group is stored as a plan value with id '_HOST_' + value of the current init value.
- plan_free - returns free units for the current resource if applicable.
- plan_value - returns plan value accessible by the current initvalue as key.

- `label` - is not used anymore, can be considered a comment.

The `initvalue` tag content is a string. If it starts with `#`, 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>
```

`<if>`

The `<if>` tag allows to include initresources or initvalues under 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.

`<values>`

The `<values>` element includes the <value> constructions inside.

`<value>`

The <value> tag defines the resource values in the plan.

Attributes:

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

Example:

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

`<special>`

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 a resouce, create the <res_RESOURCE_NAME> tag (like <res_opensrs>) inside the <special> element (you can have multiple tags insdie special section).

`<field>`

Inside `<res_RESOURCENAME>` tag, the `<field>` tag is used to define the HTML field.

Attributes:

- `type` - checkbox type;
- `name` - name of the checkbox field;
- `label` - label from hsphere_lang.properties referrring to the field caption;
- `value` - value of the checkbox;
- `checked` - equals 1 if checked.

`<include>`

Inside the `<field>` element, the `<include>` tag defines a template to be included.

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>
```


Adding Custom MS Exchange Plans into H-Sphere

(H-Sphere 2.5 and up)

Since version 2.5 H-Sphere provides integration with Microsoft Exchange mail hosting plans. The following four default MS Exchange plans are already included into H-Sphere as resources:

- Base Mail
- Gold Mail
- Platinum Mail
- Platinum Mail Pro

H-Sphere admin can also import more plans created on MS Exchange server into H-Sphere as MS Exchange hosting resources. This document is an example of a step-by-step procedure on how to do it:

1. Create custom **`msExchangePlans.xml`** file with description of your new custom MsExchange plan (it will be merged with the standard msExchangePlans.xml):

```
<plans>
   <plan type="new_plan">
      <planName>NewPlanName</planName>
      <planDescription>New MS Exchange Plan Name</planDescription>
      <planFeatures>
. . . .
         <feature>
            <featureName>some_feature</featureName>
            <featureDescription>This feature's description</featureDescription>
            <featureValue>some_value</featureValue>
         </feature>
. . . .
      </planFeatures>
   </plan>
</plans>
```

   and the **`msexchange.xml`** file that will be merged with the standard hsphere plan wizard xml (msexchange.xml):


```
<PlanWizard name="msexchange" description="planeditor.res_msexchange">
<categories>
   <category description="planeditor.other">
      <resource name="new_plan" required="1"
class="psoft.hsphere.resource.mpf.hostedexchange.HostedExchangePlan"/>
   </category>
</categories>
<resources>
   <res_bizuser>
      <mod name="new_plan">
         <initresource name="new_plan"/>
      </mod>
   </res_bizuser>
</resources>
</PlanWizard>
```

where `new_plan` is the name of the MS Exchange hosting plan you are adding. **This name must not exceed 10 symbols.**

Notes:

- Don't copy content of the existing standard HS MsExchange plan xml file.
- H-Sphere 2.5 Beta 4 and up implements a Java utility that fetches this information from MS Exchange hosting server:

  ```
  java psoft.hsphere.tools.HePlanExctractor
  ```

  This utility prints XML output with properties of all available MS Exchange hosting plans. Or, you can obtain plan features in XML form in the HostedExchange Web interface under the GetPlanDetail service.

More on how to merge xml configuration files

2. Log into the CP server as *cpanel user*.
3. Create a special location where you will assemble and build a package describing your MsExchange plan, for example, `~cpanel/shiva/custom/Packages`:

   ```
   mkdir ~cpanel/shiva/custom/Packages
   ```
4. Run package configurator:


```
cd ~cpanel/shiva/custom/Packages
java psoft.hsp.tools.PkgConfigurator --with-prefix=./ms_customization --with-xmls --with-properties
--with-sql
```

Package configurator will create the `ms_customization` directory, which includes the following structure:

`src/pkg_config/` - directory with default.properties file
`src/pkg_xmls/` - directory directory where you will place msExchangePlans.xml and msexchange.xml
`src/_pkg.xml` - file to set package name, version, build, vendor and description
`src/_SCRIPTS/_pkg.sql` - file to add the SQL query that will be executed during the package installation

5. Set the package name, version, build, vendor, description in `src/_pkg.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<pkg build="00" description="Description for test package" info="Additional information"
name="MSExchangeCustomization"
vendor="Vendor" version="00.00.00">
<xmls src="pkg_xmls/"/>
<config path="pkg_config/default.properties"/>
```

NOTE: **Package name can't contain spaces** and special characters.

6. Edit `src/_SCRIPTS/_pkg.sql`. Add the following line into it:
INSERT INTO type_name (id, name, price, description, rprice) VALUES ('resource_type_id', 'new_plan', 'RFS', 'resource_type_description', 'RFS');
where:
- 'resource_type_id' is an integer from the range 20100-21490, unique in the type_name.
- 'new_plan' is a resource type name, up to 10 symbols string without spaces.
- 'resource_type_description', string. This is your plan description.
- 'RFS' - price type, should be 'RFS' for all MS Exchange plans you add.

For example, a standart MS Exchange plan will have the following records:

hsphere=# SELECT * from type_name where description like 'Hosted Exchange%';
id | name | price | description | rprice | required | priority | ttl
------+------------+-------+--+--------+----------+----------+-----
7222 | plat_mailp | RFS | Hosted Exchange Platinum Mail Plus Plan | RFS | | |
7219 | base_mail | RFS | Hosted Exchange Base Mail Plan | RFS | | |
7221 | plat_mail | RFS | Hosted Exchange Platinum Mail Plan | RFS | | |

7220 | gold_mail | RFS | Hosted Exchange Gold Mail Plan | RFS | | |

7. Copy the prepared msExchangePlans.xml and msexchange.xml files to `ms_cutomization/src/pkg_xmls/`:

```
cp /some/location/msExchangePlans.xml ~cpanel/shiva/custom/Packages/ms_customization/src/pkg_xmls/
cp /some/location/msexchange.xml ~cpanel/shiva/custom/Packages/ms_customization/src/pkg_xmls/
```

8. Edit `src/pkg_config/default.properties`:

    1. Add properties that specify location of xml file implemented in this package:

    ```
    PLAN_WIZARDS_DIR=.
    HOSTED_EXCHANGE_PLANS = msExchangePlans.xml
    ```

    2. Return from the package directory and run package builder:

    ```
    cd ~cpanel/shiva/custom/Packages
    java psoft.hsp.tools.PkgBuilder --with-source=ms_customization
    ```

    The hsp package, for example `MSExchangeCustomization-00.00.00-01.hsp` will be created in the directory. It is ready to be *installed* on any H-Sphere 2.5 and up.

XML Manager

Related Docs: · Building Packages With XML Configuration Files

XMLManager serves for loading custom XMLs into H-Sphere and merging them with the standard, or default, H-Sphere XML configuration files. This enables to extend H-Sphere by adding new elements, like custom menu items, without changing the standard XML configuration files. This mechanism makes customization more flexible to the H-Sphere updates. Now you don't need to apply custom settings each time the standard configuration files are rewritten by new releases.

XMLManager Implementation

XMLManager is the subclass of the psoft.hsphere.util class of H-Sphere utilities. XMLManager returns either Document object or TemplateXML object.

XMLManager searches for custom XML configuration files, by the key, not by the filename:

1. in hsphere.properties;
2. in the predefined values (see `psoft.hsphere.util.PackageConfigurator`).

For example, the menu XML configuration file is set in hsphere.properties as follows:

MENU_CONFIG = /hsphere/local/home/cpanel/shiva/psoft/hsphere/menu.xml

XMLManager looks for the `MENU_CONFIG` key instead of the full pathname (`/hsphere/local/home/cpanel/shiva/psoft/hsphere/menu.xml`).

XMLManager may also process the group of XML files. It is currently implemented for XML plan wizards, where the key is compounded from the corresponding path + the name of XML file, like:

XMLManager.getXML(PLAN_WIZARDS_DIR, "unix.xml");

Example:

Consider an example of merging menu default and custom menu XMLs. For details, please refer to *Menu Customization* in Customization Guide.

1) Log in as the *cpanel user*.

2) Create custom XML file and specify its location as `CUSTOM_XML_CONFIG` in hsphere.properties (see *XML Merge Customization* in Customization Guide for details):

MENU_CONFIG = /hsphere/local/home/cpanel/shiva/psoft/hsphere/menu.xml
CUSTOM_MENU_CONFIG = /hsphere/local/home/cpanel/shiva/custom/xml/test1_menu.xml

3) To add a custom menu item, `~cpanel/shiva/custom/xml/test1_menu.xml` should look like this:

```
<?xml version="1.0"?>
<!DOCTYPE config [
 <!ELEMENT config (menus,interface)>
  <!ELEMENT menus (menu+)>
  <!ELEMENT menu (menuitem*,initmenu*)>
  <!ELEMENT menuitem (#PCDATA)>
  <!ELEMENT initmenu (#PCDATA)>
  <!ELEMENT interface (menudef+)>
  <!ELEMENT menudef (initmenu*,menuitem*)>


  <!ATTLIST menudef id CDATA #REQUIRED>

  <!ATTLIST menu name CDATA #REQUIRED>
  <!ATTLIST menu label CDATA #REQUIRED>
  <!ATTLIST menu platform_type CDATA "">
  <!ATTLIST menu resource CDATA "">
  <!ATTLIST menu defaultitem CDATA #REQUIRED>
  <!ATTLIST menu tip CDATA "">

  <!ATTLIST menuitem name CDATA #REQUIRED>
  <!ATTLIST menuitem label CDATA #REQUIRED>
  <!ATTLIST menuitem URL CDATA #REQUIRED>
  <!ATTLIST menuitem platform_type CDATA "">
  <!ATTLIST menuitem resource CDATA "">
  <!ATTLIST menuitem tip CDATA "">
  <!ATTLIST menuitem check_type CDATA "1">
  <!ATTLIST menuitem new_window CDATA "0">
```

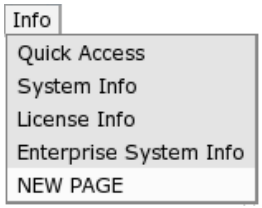


```
  <!ATTLIST initmenu name CDATA #REQUIRED>
]>

<config>
<menus>
<menu name="info" label="info.label" defaultitem="info-plans" tip="info.tip">
  <menuitem name="xyz-wow" label="NEW PAGE" URL="/newpage.html" resource="" tip="Positive Software Corporation"/>
</menu>
</menus>
</config>
```

IMPORTANT:
In the custom XML file to be merged with the default one, you must define the same DTD structure!

Also note that we don't copy the whole structure of the standard menu.xml file. The custom menu will be merged with the standard menu in the following way (the NEW PAGE item in the INFO menu):



4) To check if the merge is performed correctly, run the DOMLoader utility and check `out.xml` afterwards:

java psoft.util.xml.DOMLoader /hsphere/local/home/cpanel/shiva/psoft/hsphere/menu.xml
/hsphere/local/home/cpanel/shiva/custom/xml/test1_menu.xml > out.xml

With DOMLoader, you may merge more than two XML files:

java psoft.util.xml.DOMLoader file1.xml file2.xml ... fileN.xml > out.xml

XML Merge Processing Instructions

In your custom XML configuration files you should use the processing instructions to change or remove certain fragments in the default configuration file, instead of merging these fragments directly.

There are three major processing instructions: `<?change?>`, `<?remove?>`, and `<?merge?>`. The `merge` instruction is used by default and can be omitted.

The `<?changeattr?>` instruction is added to customize tag attributes in XML documents.

In order to change, remove, or merge an element, you need to create a well formed XML document, which describes the location of this element relative to other elements. Leave out the elements you don't want to change/remove/merge. Put the instruction before the XML element that needs to be changed/removed/merged.

Examples:

1) To remove a certain (tt) item from the the default menu configuration:

```
<?xml version="1.0"?>
<!DOCTYPE config [
 <!ELEMENT config (menus,interface)>
  <!ELEMENT menus (menu+)>
  <!ELEMENT menu (menuitem*,initmenu*)>
  <!ELEMENT menuitem (#PCDATA)>
  <!ELEMENT initmenu (#PCDATA)>
  <!ELEMENT interface (menudef+)>
  <!ELEMENT menudef (initmenu*,menuitem*)>


  <!ATTLIST menudef id CDATA #REQUIRED>

  <!ATTLIST menu name CDATA #REQUIRED>
  <!ATTLIST menu label CDATA #REQUIRED>
  <!ATTLIST menu platform_type CDATA "">
  <!ATTLIST menu resource CDATA "">
  <!ATTLIST menu defaultitem CDATA #REQUIRED>
  <!ATTLIST menu tip CDATA "">

  <!ATTLIST menuitem name CDATA #REQUIRED>
  <!ATTLIST menuitem label CDATA #REQUIRED>
  <!ATTLIST menuitem URL CDATA #REQUIRED>
  <!ATTLIST menuitem platform_type CDATA "">
```



```
  <!ATTLIST menuitem resource CDATA "">
  <!ATTLIST menuitem tip CDATA "">
  <!ATTLIST menuitem check_type CDATA "1">
  <!ATTLIST menuitem new_window CDATA "0">

  <!ATTLIST initmenu name CDATA #REQUIRED>
]>

<config>

<interface>

<menudef id="unix">
<?remove?>
        <initmenu name="tt"/>
</menudef>

</interface>

</config>
```

2) To replace the whole icon section to the one with only 3 icons:

```
<?change?>
<icons>
<icon id="changelanguage" url_param="template_name=misc/langs.html"
      rtype="" platform="" label="nomenu.language" tip="" help=""/>
<icon id="changedesign"
      url_param="template_name=misc/user_account_lf.html" rtype="" platform=""
      label="nomenu.lookfeel.alt" tip="" help=""/>
   <icon id="cgiwiz_formmail"
         url_param="template_name=quick/choice_script.html&sname=formmail"
         rtype="hosting" platform="unix"
         label="quick.quickview.cgiwiz_formmail"
         tip="quick.quickview.cgiwiz_formmail"
         help=""/>

</icons>
```

3) To remove a certain (cgiwiz_formmail) icon from the default configuration:

```
<icons>
```



```
<icon id="changelanguage" url_param="template_name=misc/langs.html"
      rtype="" platform="" label="nomenu.language" tip="" help=""/>
<icon id="changedesign"
      url_param="template_name=misc/user_account_lf.html" rtype="" platform=""
      label="nomenu.lookfeel.alt" tip="" help=""/>
<?remove?>

   <icon id="cgiwiz_formmail"
         url_param="template_name=quick/choice_script.html&sname=formmail"
         rtype="hosting" platform="unix"
         label="quick.quickview.cgiwiz_formmail"
         tip="quick.quickview.cgiwiz_formmail"
         help=""/>
</icons>
```

4) To change and add tag's attributes:

```
<?changeattr newtag="xyz" tip="My Tip"?>
<icon id="changelanguage" url_param="template_name=misc/langs.html"
      rtype="" platform="" label="nomenu.language" tip="" help=""/>
```

The resulting element will look like:

```
<icon id="changelanguage" url_param="template_name=misc/langs.html"
     rtype="" platform="" label="nomenu.language" tip="My Tip" help="" newtag="xyz"/>
```

Related Docs: · Building Packages With XML Configuration Files

CP Cron Configuration

Related Docs: · Adding Custom CP Cron Jobs

Control Panel cron configuration is represented in XML format in the `~cpanel/shiva/psoft/hsphere/cron_config.xml` file. Its location is set by the `CRON_CONFIG` parameter in `hsphere.properties`:

```
CRON_CONFIG = /hsphere/local/home/cpanel/shiva/psoft/hsphere/cron_config.xml
```

DTD Scheme | Example

Cron jobs are grouped according to their purpose. Default group is `CRON`. Cron groups are set in the `group` constructions that contain the `job` tags describing jobs in these groups:

```
<config>
  <group name="CRON" maxpriority="10" defpriority="3">
  <!-- priority of jobs within the group is ranked from 1 to 10;
       if job priority is not specified, it is 3 by default -->
  ...
    <job name="CustomCron" class="custom.cron.CustomCron" db_mark="C_CRON"
      starttime="now+15m" period="15"/>
    <!-- job starts in 15 minutes after CP start and runs every 15 minutes -->
  ...
  </group>
...
</config>
```

Group attributes:

- `name` - group name;
- `disabled` - group jobs are all disabled if `disabled="1"`. If not set, `disabled` is set to `0` (enabled);
- `maxpriority` - sets the maximum number to which job priority within a group is scaled. For example, if `maxpriority` is 15, then job priority within a group is from 1 to 15, where 1 is the least priority, and 15 is the maximum priority. If this option is not set, maximum priority for a group is 10 by default;
- `defpriority` - set the default job priority. If the job's `priority` attribute is not set, the job priority will be equal to the group's `defpriority` value.

- `defperiod` - sets the default launching period for group jobs, in minutes;
- `maxjobcount` - maximum number of jobs in the group running at a time.

Job attributes:

- `name` - job name;
- `class` - a fully qualified Java class name for the cron job (see Adding Custom CP Crons for instructions on adding custom cron job classes);
- `disabled` - if set to 1, the job is disabled; otherwise, it is enabled;
- `priority` - if set, overrides the default group priority;
- `starttime` - time when the job starts first. Format:
  - HH`:`MM - time in hours and minutes, for example, `5:15`;
  - `now` - job starts immediately after CP start/restart;
  - `now`+HH`h`MM`m` - job starts in HH hours MM minutes after CP start/restart, for example, `now+15m`, `now+2h`, `now+2h15m`.

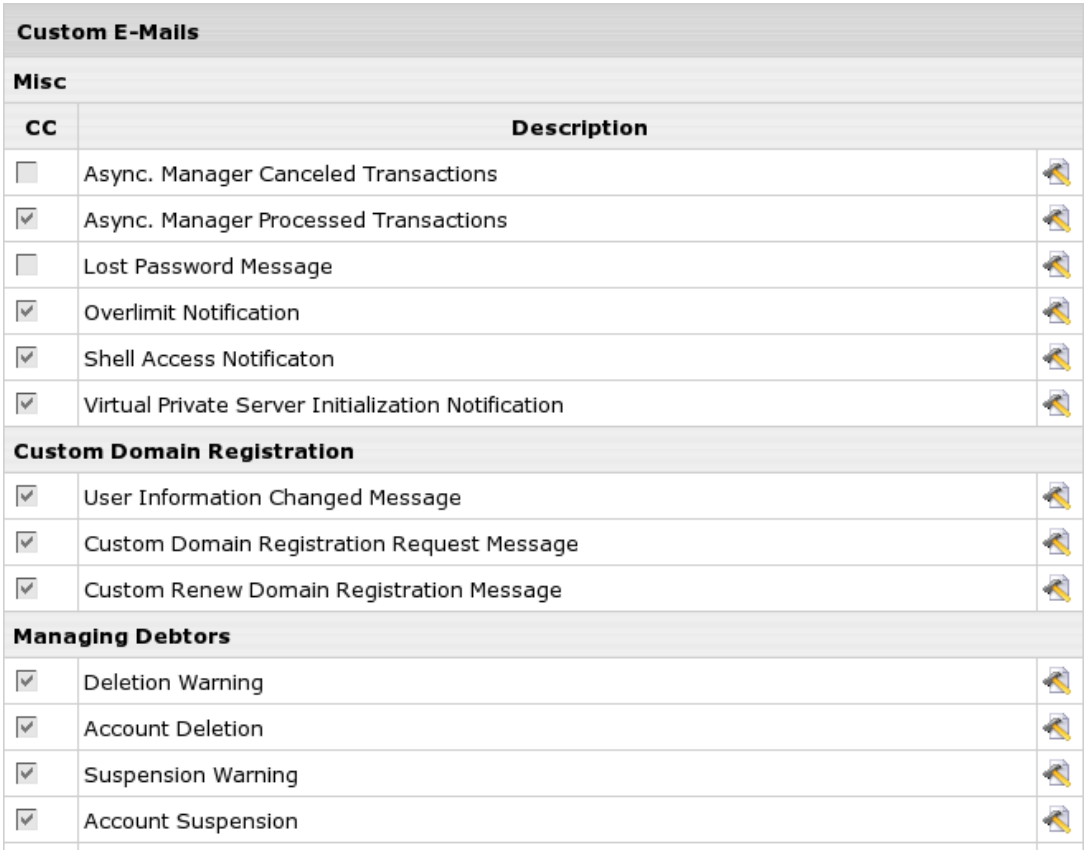
  If `starttime` is not set, the job won't start automatically.
- `period` - sets the job launching period, in minutes; if not set, the job won't run;
- `maxinstancecount` - maximum number of job instances running at a time.
- `threadcount` - maximum number of threads running at a time in a multi-thread job.
- `db_mark` - points to the job name in the H-Sphere database (value of the `name` field in the `last_start` table).
  See *CP Crons* in Sysadmin Guide for the `last_start` table description.

Related Docs: · Adding Custom CP Cron Jobs

Customizing E-Mail Notification List

Related Docs: · XML Customization With Packages · Using Variables in E-Mail Notifications

H-Sphere e-mail notifications can be customized directly in CP admin interfacer as in the **Settings->E-Mail Notifications** menu.



The list of e-mail notifications is set in the `~cpanel/shiva/psoft/hsphere/user_emails.xml` XML configuration file. It is customizable by means of H-Sphere packages.

Example | DTD Scheme

`user_emails.xml` has the following structure:

- `groups` - container for description of the groups of e-mail notifications. (In **Settings->E-Mail Notifications**, notifications are displayed in groups, according to their purpose.)
- `group` - tag for the group description. Attributes:
  - `id` - group identifier.
  - `name` - group name, corresponds to the label in *language bundles* (`~cpanel/languages/hsphere_lang.properties`).
- `emails` - container for description of e-mail notifications.
- `email` - tag with notification description. Attributes:
  - `tag` - notification tag. Corresponds to the notification's ce.TAG_NAME.title (name of the notification in the list in CP), and ce.TAG_NAME.desc (short description in the list) labels in `hsphere_lang.properties`, where TAG_NAME is the value set in the `tag` attribute of the notification in `user_emails.xml`
  - `group_id` - corresponds to the `id` attribute of the group to which this e-mail notification belongs to.
  - `template` - pathname to the corresponding *system e-mail notification template*, relative to the `~cpanel/shiva/shiva-templates/common` directory of default H-Sphere templates. For example, `template="mail/custom_registrar_registration.txt"` points to the `~cpanel/shiva/shiva-templates/common/mail/new_account.txt` template for the "welcome" message on creating a new account.
  - `massmail_applicable` - if set to `1`, this notification is available for *mass mail* delivery.
  - `admin_only` - if `admin_only="true"`, this notification is available only from the admin CP.
  - `no_cc` - if `no_cc="true"`, no copies (the BCC: field) of this e-mail notification are sent to the addresses defined in the CP admin's *Settings->Notification Recipients* menu. That corresponds to unchecking the Sent CC checkbox while editing the notification in CP.
- `subject` - tag with the subject of the e-mail message to be sent to customers.

Related Docs: · XML Customization With Packages · Using Variables in E-Mail Notifications

Using Variables in H-Sphere E-Mail Notifications

Related Docs: · XML Customization With Packages · Customizing Notification List

H-Sphere e-mail notifications can be customized directly in CP admin interfacer in the **Settings->E-Mail Notifications** menu.

This document aims at advanced customization of these messages. It contains the description of basic notification variables.

Below is the table illustrating which variables are used in which notifications. The notifications are grouped and named according to their XML configuration file, user_emails.xml. The Tag column corresponds to the names of the notifications set in the `tag` attributes in `user_emails.xml`. In the Variables column, only variables specific to that particular notifications are listed. Standard variables are variables common for all notifications. Properties and methods for variables are listed below.

The List of Notification Variables

Tag	Description	Variables
Misc		
LOST_PASSWORD	Lost Password Message	standard variables
OVERLIMIT	Overlimit Notification	suspend over_limit_res standard variables
SSH_NOTIFICATION	Shell Access Notification	

		result subject standard variables
VPS_INIT	Virtual Private Server Initialization Notification	vpsname ips ci standard variables
Custom Domain Registration		
ASYNC_CANCELED	Async. Manager Canceled Transactions	Not implemented
ASYNC_DONE	Async. Manager Processed Transactions	d toolbox
CUSTOM_REGISTRAR_CONTACT_CHANGED	User Information Changed Message	registrant tech admin billing domain_name renew_days email_days standard variables
CUSTOM_REGISTRAR_REGISTRATION	Custom Domain Registration Request Message	registrant tech admin billing domain_name renew_days email_days standard variables
CUSTOM_REGISTRAR_RENEW	Custom Renew Domain Registration Message	registrant domain_name renew_days email_days period

		standard variables
Managing Debtors		
DEBT_DEL_NOT	Deletion Warning	bill negative_date current_date suspend_date delete_date standard variables
DEBT_DEL_REASON	Account Deletion	bill negative_date current_date suspend_date delete_date standard variables
DEBT_SUSP_NOT	Suspension Warning	bill negative_date current_date suspend_date delete_date standard variables
DEBT_SUSP_REASON	Account Suspension	Not implemented
DEBT_WARN_NOTIFICATION	Outstanding Balance Notification	bill negative_date current_date suspend_date delete_date standard variables
TRIAL_SUSP_REGISTER	Trial Suspension Notification	standard variables
Welcome Messages		
FAILED_SIGNUP	Failed Signup Notification	failed_signups_q accounts

		lang
NEW_ACCOUNT	Welcome Letter	bi ci standard variables
NEW_MODERATED	Welcome Letter For Moderated Accounts	bi ci reseller_url standard variables
NEW_MODERATED_CC	Welcome Letter For Moderated Account with CC	bi ci reseller_url standard variables
TRIAL_MODERATED	Welcome Letter For Trial/Moderated Account	bi ci reseller_url standard variables
TRIAL_REGISTER	Trial Registration	standard variables
Tax Exemptions		
ACCOUNT_TEXEMPT_APPROVED	Tax Exemption Approved Notification (Live Accounts)	bi ci date standard variables
ACCOUNT_TEXEMPT_REJECTED	Tax Exemption Rejected Notification (Live Accounts)	bi ci date standard variables
MODERATED_TEXEMPT_APPROVED	Tax Exemption Approved Notification (Moderated Accounts)	bi ci date request standard variables

MODERATED_TEXEMPT_REJECTED	Tax Exemption Rejected Notification (Moderated Accounts)	bi ci date request standard variables
NEW_MODERATED_TEXEMPT	Welcome Letter (Tax Exemption)	bi ci reseller_url standard variables
Domain Registration		
DOMAIN_TRANSFER	Domain Transfer Message	domain standard variables
REGISTRAR_EXPIRED_WARN	Expired Domain Registration Notification	osrs standard variables
REGISTRAR_RENEW_WARN	Domain Registration Renew Warning	osrs standard variables
Managing Trials		
TRIAL_APPROACH_NOT	Trial Expiry Warning	current_date delete_date suspension_date trial_days_till standard variables
TRIAL_DEL_NOT	Deletion Warning	current_date delete_date suspension_date trial_days_till standard variables
TRIAL_DEL_REASON	Account Deletion	current_date delete_date suspension_date trial_days_till

		standard variables
TRIAL_SUSP_NOT	Suspension Warning	current_date delete_date suspension_date trial_days_till standard variables
Trouble Ticket System		
INTERNAL_TICKET	Internal Ticket	standard variables
Accounting		
ACCOUNTING_ERROR	Accounting Error letter	error date stack error_subject standard variables
INVOICE	Order Confirmation	entries taxes subtotal total reseller_url standard variables
MONEY_BACK	Money Back Notification	standard variables
Suspend/Resume		
RESUME	Account Resumed Notification	standard variables
SUSPEND	Account Suspended Notification	reason standard variables

Standard Variables

Variable	Description
account	The instance of the Account object
user	The instance of the User object
plan	The instance of the Plan object
toolbox	The instance of the Toolbox object
config	Values from hsphere.properties
settings	Current reseller settings
lang	The Lang object

Special Variables

Variable	Description
suspend	The date when account will be suspended
over_limit_res	The list of overlimited resources: Traffic, Quota, SummaryQuota
result	The result of the request (OK, REFUSED, DISABLED), used only in SSH_NOTIFICATION
subject	The result description, used in SSH_NOTIFICATION
vpsname	VPS server name
ips	List of VPS server IPs
d	The instance of the AsyncDescriptor object, used only in ASYNC_DONE
registrant	The same as ContactInfo
tech	Tech info
admin	Admin info

billing	Registrar Billing Info
domain_name	Domain name
renew_days	Renew domains that many days before domain expiration
email_days	Warn users about domain expiration that many days in advance
period	Renewal period, in years
bill	The instance of the Bill object
negative_date	Outstanding balance date
current_date	Current date
suspend_date	The date when account will be suspended
delete_date	The date when account will be deleted
failed_signups_q	Failed signups counter, only for FAILED_SIGNUP
accounts	List of accounts with failed signup, only for FAILED_SIGNUP
bi	The instance of the BillingInfo object
ci	The instance of the ContactInfo object
date	Current date
request	Fake request object
reseller_url	Reseller URL
suspension_date	The date when the account will be suspended
trial_days_till	Days left till the end of the trial period
error	Error message
stack	Java stack trace
error_subject	Error's short description
entries	The list of bill entries

taxes	List of taxes
subtotal	The total, without taxes
total	The total, with taxes
reason	Reason for suspension, used only in SUSPEND

Properties and Methods

Many variables used in e-mail notifications are instances of H-Sphere objects. Below is the table with the description of methods and properties of respective objects (in bold).

Property/Method	Description
Account	
id	Account ID
description	Description
periodId	Billing period ID
bi	The BillingInfo object
ci	The ContactInfo object
bill	The Bill object
plan	The Plan object
planId	Plan ID
receive_invoice	Flag indicating if the invoice is received
suspend_reason	Reason for suspension
password	Password
login	Login
trial_time_left	Time left from trial account creation

p_end	Billing period end date
exhaustion_date	Exhaustion date
created	Account creation date
User	
login	Login
password	Password
reseller_id	Reseller ID
reseller_url	Reseller's CP URL
reseller_context_url	Reseller Context URL
prefix	Prefix for databases
isdemo	Flag indicating that the account is in demo mode
Plan	
id	Plan ID
description	Description
reseller_id	Reseller ID
disabled	Flag indicating that plan is disabled
isPromocodeApplicable	Flag indicating if promocode is available
type	Plan type
Toolbox	
getInvoice(modId)	Returns an Invoice object
calculateTaxes(total[,bi_id])	Calculates taxes
currency(value)	Returns a string with currency representation
numberToCurLocale(value,useGrouping)	Converts a number to current locale

getPaymentLink(gateway, sbalance, prefix, account, description)	Returns payment link
int2ext(ip)	Converts internal IP to external
lt(val1,val2)	Compares two variables. Returns 1 if val<val2, else 0
sub(val1,val2)	Subtracts val2 from val1
mul(val1,val2)	Multiplies val1 and val2
taxes	The list of taxes
displayBalance(balance)	Returns a string with the balance
now	Current date
date	Current date
Traffic	
traffic	Current traffic
size	Traffic limit
tt_type	Traffic type
text_traffic	Text representation of traffic amount
info	Current traffic status
start_date	Date when traffic calculation is started
Quota	
limitMb	Quota size, in MB
limitFiles	Limit of used files
usedMb	Disk space now in use, in MB
usedFiles	Used files counter
info	Current quota status
start_date	Date when disk usage calculation is started

SummaryQuota	
limitMb	Disk usage limit
usedMb	Current disk usage, in MB
lastDayUsedMb	The last day usage, in MB
info	Current disk usage status
start_date	Date when disk usage calculation is started
short_start_date	Disk usage calculation starting date, short format
ContactInfo	
first_name	First Name
last_name	Last Name
org_name	Company Name
address1	First address
address2	Second address
address3	Third address
city	City
state	State
country	Country
postal_code	ZIP
phone	Phone Number
fax	Fax number
email	E-Mail address
BillingInfo	
id	Billing contact info ID

bi_id	Bill ID
name	Name
first_name	First Name
last_name	Last Name
org_name	Company Name
address1	First address
address2	Second address
city	City
state	State (US)
state2	Other state (non-US)
country	Country
postal_code	ZIP
phone	Phone number
fax	Fax number
email	E-mail address
reason	Reason for suspension
type	Payment type: `CC` - credit card, `CHECK` - check, `TRIAL` - trial account.
pi	The PaymentInfo object
exemption_code	Tax exemption code
negative_date	Outstanding balance date
PaymentInfo	
number	CC number

hNumber	Hidden number
cvv_checked	Flag indicating if CVV is checked
exp	CC expiration date
exp_year	CC expiration year
exp_month	CC expiration month
exp_day	CC expiration day
start_date	CC creation date
start_year	CC creation year
start_month	CC creation month
start_day	CC creation day
issue_no	Issue number
name	CC name
expired	Flag indicating that CC is expiried
info	CC info
type	CC type
fatts	Failed Charge Attempts Counter
fatts_info	Failed Charge Attempts Info
is_fatts_checked	Flag indicating that Fatts control is activated
AsyncDescriptor	
user_id	User ID
user	The User object
rid	Resource ID
description	Description

start_date	Start date
last_check	Last check
state	State
max_delay	Maximum delay
interval	Interval
error	Error message
error_code	Error code
Bill	
id	Bill ID
opened	Date when the billing period was opened
closed	Date when the billing period was closed
entries	List of billing entries
description	Description
desc_plan	Plan description
desc_short	Short description
start_balance	Start of the billing period
end_balance	End of the billing period
debits	Debit
credits	Credit
change	Difference
amount	Amount
subamount	Amount without taxes
balance	Balance

credit	Credit
customCredit	Custom credit
negativeDate	Outstanding balance
format_opened	Formatted date when the billing period was opened
format_closed	Formatted date when the billing period was closed
to_pay	Negative balance
taxes	List of taxes
total_taxes	Taxes, total
BillEntry	
id	Billing entry ID
bill_id	Billing period ID
description	Description
is_credit	Flag indicating if credit is added
tax	Tax
amount	Amount
balance	Balance
note	Note
type	Payment type
canceled	Flag indicating that billing entry is canceled
billing_info_id	Billing Info ID
date	Date when the billing entry was added
format_date	Formatted date when the billing entry was added
Invoice	

localized_total	The total in local currency
total	The total
subtotal	The total without taxes
entries	Billing entries
taxes	The list of taxes

Examples of Usage

1. Labels from language bundles

Labels from the `hsphere_lang.properties` *language bundles* are called by means the `lang` variable.

For example, the `massmail.welcome_login` label is called:

```
${lang.massmail.welcome_login}
```

In more complex cases, labels may contain arguments within their text, for example:

```
massmail.welcome_header = Dear {0} {1},
```

where the arguments `{0}` and `{1}` stand for a customer's first name and last name, respectively.

To call such labels with arguments:

```
${lang.massmail.welcome_header(account.ci.first_name,account.ci.last_name)}
```

Please refer to the description of account and ci.

2. Getting credit card information

The pi variable returns information about user credit card. For example, to get credit card number:

${account.bi.pi.number}

Related Docs: · XML Customization With Packages · Customizing Notification List

Creating Unix-Hosted Resources

A Unix-hosted resource is created as follows:

1. ***Login as cpanel user*** **to CP server.**
2. **Create a class that implements this resource.**

   Create a new H-Sphere resource class in the `~cpanel/shiva/psoft/hsphere/resource` directory. For example:

```
package psoft.hsphere.resource;
import psoft.hsphere.*;
import java.util.Collection;

public class MyResource extends Resource {
    public MyResource(int type, Collection init) throws Exception {
        super(type, init);
        ...........
    }
    public MyResource(ResourceId rid) throws Exception {
        super(rid);
        ...........
    }
}
```

   The class must have 2 constructors, as illustrated in the above example.

   1. The first constructor, `MyResource(int type, Collection init)`, is used on the resource creation and its `init` argument contains the list of the String class objects. The `MyResource` method's arguments starting from the third are parameters passed from the HTML form. For example:

      ```
      <assign result=parent.addChild(request.resource_name,"", db_id, user_id)>
      ```

      Here, `db_id` and `user_id` are parameters taken from the HTML form.

      In this case, the values of 'db_id' and 'user_id' are passed to constructor as Collection of Strings.
   2. The second constructor, `MyResource(ResourceId rid)`, is used when the user (owner of the resource) is logged in and `ClassLoader` loads the resource.

Also, the resource may overwrite the `initDone()` method called upon the resource creation. `initDone()` is designed to save all additional parameters into the database and to execute additional scripts.

The resource may also overwrite the `delete()` method called upon the recource deletion. This method is used to delete all additional database records related to this resource.

If the resource creates system resources (directories, files, etc.) on a certain host, the resource should implement the HostDependentResource interface:

```
package psoft.hsphere.resource;

public interface HostDependentResource {
    public boolean canBeMovedTo(long newHostId) throws Exception;
    public void physicalCreate(long targetHostId) throws Exception;
    public void physicalDelete(long targetHostId) throws Exception;
    public void setHostId(long newHostId) throws Exception;
    public long getHostId() throws Exception;
}
```

In case of the `HostDependentResource` implementation, the method `physicalCreate(long targetHostId)` must be called from the method `initDone()`, and the method `physicalDelete(long targetHostId)` from the method `delete()`. The methods `physicalCreate(long targetHostId)` and `physicalDelete(long targetHostId)` call scripts to create and delete system resources, respectfully.

The method `canBeMovedTo(long newHostId)` should return `true` if physical components of the resource (files, directories, etc) can be moved to or created on the host. The methods `setHostId(long newHostId)` and `getHostId()` respectfully set and get id of the host where resource physical components are located.

General order of calling methods during the first resource creation is as follows:

```
MyResource(int type, Collection init)
public void initDone()
public void physicalCreate(long targetHostId)
```

The resource parameter values predefined in the `<plan_name>.xml` file can be retrieved by calling the `getPlanValue(String key)` method defined in `Resource.class`.

For example, in `unix.xml`, the `myresource` resource is described with the `res_myresource` construction with the block of predefined values:

```
<res_myresource>
```



```
...
  <values>
    <value name="some_key">some value</value>
    ...
  </values>
</res_myresource>
```

Then, to get the value for `some_key`, the `getPlanValue` method is used as follows:

```
<assign value=myresource.getPlanValue("some_key")>
```

In order to retrieve the resource values, `FM_` methods should be implemented to be called from the HTML form. `FM_` method, with its name having the `FM_` prefix, is a method that returns a value of the `TemplateModel` class.

If you call an `FM_` method from a template, the Freemarker code will look like:

```
<assign multiply=myresource.multiply(a, b)>
```

Here, `myresource` is the resource name in the plan, and `multiply` refers to the `FM_multiply(a, b)` method, where `a` and `b` are the method's arguments.

In this case, the `TemplateModel FM_multiply(int x, int y)` method of `MyResource` class will be called.

3. **Insert resource information into the H-Sphere database.**

Information must be added to the `type_name` table:

```
   Column    |          Type          | Modifiers
-------------+------------------------+-----------
id           | integer                | not null
name         | character varying(40)  | not null
price        | character varying(5)   |
description  | character varying(100) |
rprice       | character varying(5)   |
required     | smallint               |
priority     | smallint               |
ttl          | integer                |
```

Index: `type_name_name`

Primary key: `type_name_pkey`

Field description:

- `id` - unique resource type id
- `name` - unique resource name
- `price` - flags:
  - `M` (initial value must be set)
  - `R` (recurrent)
  - `U` (usage)
  - `F` (initial value is optional)
  - `S` (setup)
- `description` - resource description
- `rprice` - reseller flags
  - `R` (recurrent)
  - `U` (usage)
  - `F` (initial value is optional)
  - `S` (setup)
- `required` - flag indicating if this resource is required for all accounts
- `priority` - must be 0
- `ttl` - resource deletion time (must not be set initially)

To add a resource, insert the resource data into the `type_name` table, like this:

```
INSERT INTO type_name (id,name,description,required,priority) VALUES (119, 'my_resource', 'My Resource',0,0);
```

4. ## Include the resource to the plan XML configuration file.

The resource should be included into XML configuration files for each type of plans where this resource is to be enabled (for example, `unix.xml` for Unix plan). These files are located in the `~cpanel/shiva/psoft/hsphere/plan/wizard/xml/` directory.

You need to add the `resource` tags with the new resource wherever necessary according to the guide on creating XML plan wizards.

For example:

```
<categories>
.....................
<category description="planeditor.myresource">
    ....................................
```



```
    <resource name="my_resource" class="psoft.hsphere.resource.MyResource" adminonly="1"/>
    ....................................
</category>
....................
</categories>
```

To make the resource available automatically upon accounts' creation, add the `res_RESOURCE_NAME` construction to the plan XML file, where `RESOURCE_NAME` is the value set in the `name` attribute of the `resource` tag for the given resource.

Make sure you also set the interface text label set as the `description` attribute of the `category` tag. This is done in the *custom* `hsphere_lang.properties` file. For the label in the above example, the label is set as follows:

```
planeditor.myresource = My Resource
```

5. **Implement the resource in CP Web interface.**

1) Create templates for this resource.

Create templates according to the *template* procedure. See the example below for the templates you may need to add.

2) Add a new menu item for this resource in CP menu.

Customize CP menu and add a new menu item to the custom menu.xml, for example:

```
.................
<menudef id="unix">
.................
    <menuitem name="myresource" label="myresource.label"
                URL="design/myresource.html" resource=""
                tip="myresource.tip"/>
.................
</menudef>
.................
```

Then, add labels `myresource.label` and `myresource.tip` to the *custom* `menu.properties` file and run language bundle compiler:
`[]$ java psoft.hsphere.LangBundlesCompiler`

Important: Starting with H-Sphere 3.0 RC 1, `menu.properties` and `messages.properties` become deprecated, and all labels are merged into a single `hsphere_lang.properties` for each language!

6. **Create scripts for the resource's physical implementation.**

   You must be logged in as root to create these scripts in the `/hsphere/shared/scripts/` directory. See the example below on how you may create these scripts.

IMPORTANT:
Make changes to **custom** templates, interface text bundles and XML files, not to the default ones! Otherwise, you will lose your settings with the next H-Sphere upgrade. Please refer to *Customization Guide* for details.

Example

Below is a detailed example of how to create and add a new resource to H-Sphere.

1. *Login as cpanel user* to CP server.
2. Create a resource class:

```
package psoft.hsphere.resource;

import java.util.*;

import psoft.hsphere.*;

public class MyResource extends Resource implements HostDependentResource {

    public MyResource(int type, Collection init) throws Exception {
        super(type, init);
    }

    public MyResource(ResourceId rid) throws Exception {
        super(rid);
    }

    /*In this case the resource can be moved to any host*/
    public boolean canBeMovedTo(long newHostId) throws Exception {
        return true;
    }

    /* This method is started when the resource is created */
    public void initDone() throws Exception {
           physicalCreate(Long.parseLong(recursiveGet("host_id").toString()));
    }
```



```
    /*
    * Deletes persistant representation of the resource as well as its system/physical representation
    * @throws Exception
    */
    public void delete() throws Exception {
        physicalDelete(Long.parseLong(recursiveGet("host_id").toString()));
    }

    /*This method implements method of HostDependentResource interface.
    * In this case method collects the information about resource
    * and runs the 'myresource-init' script with collected
    * information as arguments.
    */
    public void physicalCreate(long targetHostId) throws Exception {

     //Unix-based realization:
     List l = new ArrayList();
        l.add(recursiveGet("login").toString());
        l.add(recursiveGet("dir").toString());
        HostEntry he = HostManager.getHost(targetHostId);
        he.exec("myresource-init", l);

     //Win-based realization:
//     WinHostEntry he = (WinHostEntry) HostManager.getHost(targetHostId);
//     boolean isSOAP = WinService.isSOAPSupport();
//     if (isSoap) {
//             SOAPEnvelope envelope = he.invokeMethod("create", new String[][] {
//         {"resourcename", "myresource"},
//         {"username", recursiveGet("login").toString()},
//         {"directory", recursiveGet("dir").toString()}}
//             );
//         if (envelope == null) {
//             throw new HSUserException("The error invoked during SOAP calling");
//         }
//     } else {
//         throw new HSUserException("SOAP is not supported");
//     }

    }

    /*This method implements method of HostDependentResource interface.
    * In this case method collects the information about resource
```



```
    * and executes the 'myresource-init' script with collected
    * information as arguments in order to physically delete resource.
    */
    public void physicalDelete(long targetHostId) throws Exception {

     //Unix-based realisation:
        List l = new ArrayList();
        l.add(recursiveGet("login").toString());
        l.add(recursiveGet("dir").toString());
        HostEntry he = HostManager.getHost(targetHostId);
        he.exec("myresource-delete", l);

     //Win-based realisation:
//     WinHostEntry he = (WinHostEntry) HostManager.getHost(targetHostId);
//     boolean isSOAP = WinService.isSOAPSupport();
//     if (isSoap) {
//              SOAPEnvelope envelope = he.invokeMethod("delete", new String[][] {
//         {"resourcename", "myresource"},
//         {"username", recursiveGet("login").toString()},
//         {"directory", recursiveGet("dir").toString()}}
//             );
//         if (envelope == null) {
//             throw new HSUserException("The error invoked during SOAP calling");
//         }
//     } else {
//         throw new HSUserException("SOAP is not supported");
//     }

    }

    /*This method implements method of HostDependentResource interface.
    * id of this resource host cannot be set
    */
    public void setHostId(long newHostId) throws Exception {
        return;
    }

    /*This method implements method of HostDependentResource interface
    * and returns value of "hos_id" key in resource tree.
    */
    public long getHostId() throws Exception {
        return Long.parseLong(recursiveGet("host_id").toString());
    }
```



```
}
```

3. Insert the resource info into the system database:

```
[]$ psql hsphere
hsphere=# INSERT INTO type_name (id, name, description, required, priority)
hsphere-# VALUES (777, 'my_resource', 'My Resource', 0, 0);
```

4. Add the new resource to plan XML config files.

   For example, `~cpanel/shiva/psoft/hsphere/plan/wizard/xml/unix.xml` for Unix plan type:

```
<categories>
.....................
<category description="planeditor.myresource">
    ....................................
    <resource name="my_resource" required="0" noprice="1"
                class="psoft.hsphere.resource.MyResource"/>
    ....................................
</category>
.....................
</categories>
```

   Note: Resource will be created automatically with accounts if the `<res_my_resource> </res_my_resource>` tag is defined in unix.xml

   Then, set interface text label for the new resource:

   Edit `~cpanel/shiva/custom/bundles/hsphere_lang.properties`:

```
......
planeditor.myresource = My Resource
......
```

   Then, run language bundle compiler:

```
[]$ java psoft.hsphere.LangBundlesCompiler
```

5. Implement the resource in CP Web interface:

   1) create resource template files:

   Create `~cpanel/shiva/custom/templates/common/design/myresource.html`:


```
<assign template="design/myresource.html">
<include "functions">
<call draw_menu("myresource")>
<include "control/unixuser/myresource.html">
<call bottom()>
```

Create `~cpanel/shiva/custom/templates/common/unixuser/myresource.html`:

```
<call draw_tooltip()>
<call draw_table_header("2", lang.domain.editincluded.ftp)>
<tr>
    <td><call draw_label("My Resource")></td>
    <td><call onoff(account.getChild("unixuser"), "my_resource")></td>
</tr>
<call draw_table_footer()>
```

2) add a link to this resource in CP menu

Add a new menu item in `~cpanel/shiva/custom/xml/menu.xml`:

```
.....................
<menudef id="unix">
.....................
    <menuitem name="myresource" label="myresource.label"
                URL="design/myresource.html" resource=""
                tip="myresource.tip"/>
.....................
</menudef>
.....................
```

Then, add labels `myresource.label` and `myresource.tip` to `~cpanel/shiva/psoft/hsphere/lang/menu.properties` and run language bundle compiler:

```
[]$ java psoft.hsphere.LangBundlesCompiler
```

Important: Starting with H-Sphere 3.0 RC 1, `menu.properties` and `messages.properties` become deprecated, and all labels are merged into a single `hsphere_lang.properties` for each language!

6. Create scripts for physical implementation of the resource.

Note: You must be logged in as root.

Create the file `hsphere/shared/scripts/myresource-init`:

```
#!/bin/sh
cd $2;
mkdir myresource;
exit 0
```

Create the file `hsphere/shared/scripts/myresource-delete`:

```
#!/bin/sh
cd $2;
rm -rf myresource;
exit 0
```


Adding Web Payment Systems

(deprecated since version 2.5)

Related Docs: · Web Payment SDK (HS 2.5+)

To add a web payment system to H-Sphere, you need to:

1. Create Java servlet
2. Create .html templates
3. Create .sbm template
4. Add new web payment servlet to web.xml
5. Add new web payment system to merchants.xml

1. Create Java Servlet

Java Servlet is used to redirect users to a web payment system's payment page or receive the parameters from a web payment system and check them before adding payment to users' balance. Use the following example to create a java servlet for your custom web payment system:

```
package psoft.hsphere.payment;

import psoft.hsphere.Session;
import psoft.hsphere.resource.admin.MerchantManager;
import psoft.hsphere.resource.tt.Ticket;
import psoft.util.freemarker.TemplateString;
import psoft.util.freemarker.TemplateMap;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.StringTokenizer;
```



```
import freemarker.template.Template;
import freemarker.template.SimpleHash;


public class ExampleWebPaymentServlet  extends ExternalPayServlet {
    private static final String GATEWAY = "ExampleWebPayment";


    //The below method is called when web payment system or H-Sphere
    //calls this ExampleWebPayment servlet.
    public void service(HttpServletRequest request,
                        HttpServletResponse response)
        throws ServletException, IOException
    {
        psoft.hsphere.CP.getCP().setConfig();
        //If the servlet is called by H-Sphere
        //(to redirect customers to the payment page on the Web Payment System
        //side), the request value will contain paramter "action"
        String action = request.getParameter("action");
        if ("redirect".equals(action)) {
            //in this case the code redirects users to
            //the web payment system page
            getProcessForm(request, response);
        } else {
            //in this case, the request parameters are checked
            //and if they are correct, the user's balance is debited
            processPayment(request, response);
        }
    }

    protected void getProcessForm(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException
    {
        try {
            response.setContentType("text/html");
            Session.save();

            //The cp_url, trnas_id, amount and description parameters
            //are received from the request
            String cpUrl = request.getParameter("cp_url");
            if (cpUrl == null ||
                        "".equals(cpUrl))
            {
```



```
            throw new Exception("Empty cp_url parameter");
        }
        String trDesc = request.getParameter("trans_id");
        if ((trDesc == null)
                || "".equals(trDesc)) {
            throw new Exception("Empty transaction ID");
        }
        String amount = request.getParameter("amount");
        if ((amount == null)
                || "".equals(amount)) {
            throw new Exception("Empty amount for trans ID:" + trDesc);
        }

        String description = request.getParameter("description");
        setResellerId(trDesc);
        //the below instruction extracts the parameters that were set while
        //configuring web payment system (e.g. merchant ID, login etc.)
        HashMap values = MerchantManager.getProcessorSettings(GATEWAY);
        //the below instruction checks if the web payement system is active
        if (!"1".equals(values.get("enabled"))) {
            throw new Exception("ExampleWebPayment is not available");
        }
        //the below set of instructions extracts expample.sbm template
        String template_name = "submit/billing/example.sbm";
        Template template = Session.getTemplate(template_name);

        if (null == template) {
            processEmptyResponse(request, response, "Template " + template_name + " is not exist");
            return;
        }
        SimpleHash root = new SimpleHash();
        root.put("trDesc", new TemplateString(trDesc));
        root.put("values", new TemplateMap(values));
        root.put("amount", new TemplateString(amount));
        root.put("servlet_url", new TemplateString(cpUrl + SERVLETPATH
                + values.get("servlet")));
        root.put("description", new TemplateString(description));
        root.put("invoice", new TemplateString(
                String.valueOf(Session.getNewIdAsLong()) + "-" + trDesc));
        template.process(root, response.getWriter());
    } catch (Exception e) {
        log.warn("Unable to process transaction", e);
        Ticket.create(e, null);
```



```
            processEmptyResponse(request, response, "The payment can't be processed: "+e.getMessage());
        } finally {
            try {
                Session.restore();
            } catch (Exception ex) {
            }
        }
    }

    protected void processPayment(HttpServletRequest request,
                                  HttpServletResponse response)
        throws ServletException, IOException
    {
        //several specific hsphere variables were included in the
        //request sent earlier by getProcessForm(...) method
        //and example.sbm template (described above)
        //Now these values should be extracted trom the received request.
        //This way the invoice value as well as some other values
        //described below should be initialized.
        //invoice value should have the following format:
        // hsphere_generated_transaction_id-acc-accountID
        // hsphere_generated_transaction_id-sig-signupID
        // hsphere_generated_transaction_id-mail-accountID
        String invoice = request.getParameter("parameter_name_in_the_payment_system_request");
        //the below instruction gets the amount value from the request
        double amount =Double.parseDouble(request.getParameter("parameter_name_in_the_payment_system_request"));
        //here using the request you should check if the payment was sucessful
        //if the payment was successful, run the following set of instructions:
        StringTokenizer tokenizer = new StringTokenizer(invoice, "-");
        String transID = "";
        String accInfo = "";
        short success = 0;
        try {
            try {
                Session.save();
                tokenizer = new StringTokenizer(invoice, "-");
                transID = tokenizer.nextToken().trim();
                accInfo = invoice.substring(transID.length()+1);
                setResellerId(accInfo);

                //the below instruction adds payment to the user's billing balance
                setPayment(amount, transID, invoice, GATEWAY);
                success = 1;
```



```
            } catch (Exception e) {
                log.warn("Unable to add payment to the ", e);
            } finally {
                Session.restore();
            }
        } catch (Exception e) {
            log.warn("Unable to process transaction", e);
        }
        printResultPage(response, success);
    }

    private void printResultPage(HttpServletResponse response,
                                 int status)
            throws IOException
    {
        response.setContentType("text/html");
        PrintWriter out = new PrintWriter(response.getWriter());
        out.println("<html><body>");
        switch (status) {
            case 0 :
                out.println("Unable to process transaction");
                break;
            case 1 :
                out.println("Transaction has been completed successfully.");
                break;
            default :
                out.println("Unable to process transaction");
                break;
        }
        out.println("</body></html>");
        out.close();
    }

}
```

When you create the servlet put it into the `/home/cpanel/shiva/psoft/hsphere/payment` directory.

2. Create .html Templates

You need to create three HTML templates for the payment system settings page following the rules laid out in the *Template Customization* manual.

1) Create template frame

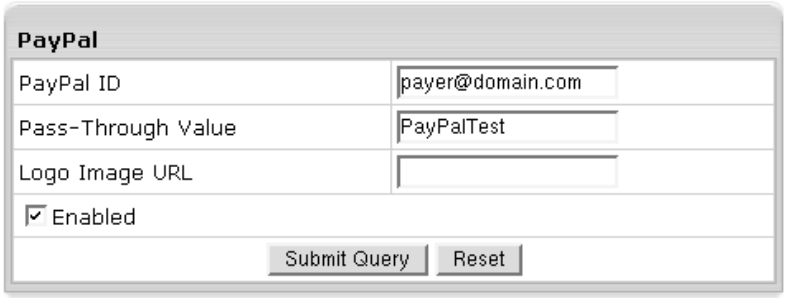
Template frame includes template controls and defines the "template" value. This is an example of a template frame:

```
<!-- BEGIN ./admin/merchant/WebPayment.html -->
<assign template="admin/merchant/WebPayment.html">
<include "functions">
<call draw_menu("merchant_view")>
<include "control/admin/merchant/WebPayment.html">
<call bottom()>
<!-- END ./admin/merchant/WebPayment.html -->
```

When you are through, put the template frame into: `/shiva-templates/common/admin/merchant/`

2) Create template controls

A control template collects web payemnt system configuration data, for example:



This is an example of a control template:

```
<!-- BEGIN ./common/control/admin/merchant/WebPayment.html -->
<assign mm=account.getChild("merchant_manager")>
<call draw_tooltip()>
<table width="70%" border="0" cellspacing="0"
```



```
cellpadding="0" align="CENTER" bgcolor="${design.color("border_color")}">
<tr><td>
<table width="100%" border="0" cellspacing="1"
cellpadding="3" align="CENTER">
<tr bgcolor="${HEADER_COLOR}">
<td colspan=2><call draw_header(lang.admin.webpayment.header)></td>
</tr>
<form action="${config.CLIENT_CP_URL}" method="POST"accept-charset="UTF-8">
<input type="hidden" name="template_name"value="submit/admin/merchant/WebPayment.sbm">
<input type="hidden" name="ftemplate" value="${template}">
<input type="hidden" name="stemplate" value="admin/merchant/view.html">
<tr bgcolor="${LIGHT_STRIP}">
<td><call draw_label(lang.admin.webpayment.value_1)></td>
<td><input type="text" name="value_1" value="${settings.WebPayment_VALUE_1}"></td>
</tr>
<td><call draw_label(lang.admin.webpayment.value_2)></td>
<td><input type="text" name="value_1" value="${settings.WebPayment_VALUE_2}"></td>
</tr>
...
...
...
<td><call draw_label(lang.admin.webpayment.value_N)></td>
<td><input type="text" name="value_N" value="${settings.WebPayment_VALUE_N}"></td>
</tr>

<if request.enabled == "1">
<assign venabled="CHECKED"><else><assignvenabled="">
</if>
<tr bgcolor="${LIGHT_STRIP}">
<td colspan="2"><input type=checkboxname="enabled" ${venabled}>
<calldraw_label(lang.label.enabled)>
</td>
</tr>
<tr bgcolor="${LIGHT_STRIP}">
<td colspan="2" align="center">
<input type="submit">
</td>
</tr>
</form>
</table>
</td>
</tr>
</table>
```



```
<!-- END ./common/control/admin/merchant/WebPayment.html -->
```

3) Create submit template

The submit tempalte posts web payment configuration data into H-Sphere database. This is the example of a submit template:

```
 <assign admin = account.getChild("admin")>
  <assign mm=account.getChild("merchant_manager")>
  <if request.enabled!="">
  <assign enabled="1">
  <else>
  <assign enabled="">
  </if>
  <if admin>
  <assign res=mm.webprocessorparams("WebProcessor", "VALUE_1", request.value_1,
  "VALUE_2", request.value_2,...,"VALUE_N", request.value_n, "enabled",
  enabled)>
  <if res.status=="OK">
    <include request.stemplate>
  <else>
    <assign res=session.addMessage(res.msg)>
    <include request.ftemplate>
  </if>
  </if>
```

When you are through, put the submit template into `/shiva-templates/common/submit/admin/merchant/`

3. Create .sbm Template

.sbm template redirects customers to the web payment payment page when users decide to pay with your custom web payment system. It should be put to `/common/submit/billing`.
Follow this example:

```
<HTML>
```



```
<HEAD>
</HEAD>
<BODY BGCOLOR="${design.color("bgcolor")}">
<assign tmp_action = "">https://webpaymentsystem_purchase_page_url">
<FORM name="login" action="${tmp_action}" method="post" accept-charset="UTF-8">
<input type="hidden" name="value_1" value="${values.VALUE_1}">
<input type="hidden" name="value_2" value="${values.VALUE_2}">
...
...
<input type="hidden" name="value_N" value="${values.VALUE_N}">
<input type="hidden" name="desc" value="${description}">
<input type="hidden" name="amount" value="${amount}">
</FORM>

<SCRIPT language="javascript">
window.document.forms['login'].submit();
</SCRIPT>
</BODY>
</HTML>
```

4. Add New Web Payment Servlet To web.xml

Add a record about the new servlet to `shiva/psoft/hsphere/WEB-INF/web.xml`, for example:

```
<servlet>
<servlet-name>psoft.hsphere.payment.WebPaymentServlet</servlet-name>
<servlet-class>psoft.hsphere.payment.WebPaymentServlet</servlet-class>
<init-param>
    <param-name>debug</param-name>
    <param-value>1</param-value>
</init-param>
<init-param>
    <param-name>listings</param-name>
    <param-value>true</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>psoft.hsphere.payment.WebPaymentServlet</servlet-name>
```



```
    <url-pattern>/psoft.hsphere.payment.WebPaymentServlet</url-pattern>
</servlet-mapping>
```

Here, initial parameters for the servlet are set in the `init-param` tags. `servlet-mapping` specifies the servlet pathname in the URL by which the servlet is called on the Web, e.g., in the example above, `http://cp.domain.com:8080/psoft.hsphere.payment.WebPaymentServlet`.

5. Add New Web Payment System To merchants.xml

Add a new record into `shiva/psoft/hsphere/merchants.xml` within the `<processors>` tag, for example:

```
<processor name="WebPayment" description="label.webpayment"
template="admin/merchant/WebPayment.html" servlet="WebPaymentServlet">
        <value name="value_1"/>
        <value name="value_2"/>
...
...
        <value name="value_N"/>
</processor>
```

Here, `value_1-value_N` are the fields for values to be stored in the H-Sphere database.

Please note that you don't specify the full servlet name (`psoft.hsphere.payments.WebPaymentServlet`) in the servlet attribute.

Related Docs: · Web Payment SDK (HS 2.5+)

Web Payment SDK

(version 2.5 and higher)

H-Sphere Web Payment SDK is a development tool to integrate *Web payment systems* as *installable H-Sphere packages (.hsp)*. The SDK allows developing and testing Web payment packages outside H-Sphere.

To use Web payment SDK, make sure you have the following software installed:

- Java 1.3 or higher
- Apache Ant 1.5 or higher - a Java-based build tool
- JUnit - a framework to write repeatable tests

Download Latest Web Payment SDK | Online Documentation

Domain Registrar SDK

H-Sphere Registrar SDK is a toolkit to integrate *domain registrars* as *installable H-Sphere packages (.hsp)*. The SDK allows developing and testing gateway packages outside H-Sphere.

To use Resistrar SDK, make sure you have the following software installed:

- Java 1.3 or higher
- Apache Ant 1.6.1 or higher - a Java-based build tool
- JUnit - a framework to write repeatable tests

Download Latest Registrar SDK | **Online Documentation**

Adding Custom CP Cron Jobs

Related Docs: · CP Cron Configuration

CP cron jobs are internal H-Sphere Java utilities that perform regular tasks such as accounting, trouble ticket management, etc.

It is possible to manage crons in XML configuration files and to add custom cron jobs.

To create a custom cron job:

1. **Log in as the *cpanel user*.**
2. **Create the cron job class**

   Extend the abstract class psoft.hsphere.background.BackgroundJob and implement the method
   `protected abstract void processJob() throws Exception`
3. **Add the custom cron job class to CLASSPATH**

   CLASSPATH is set in the `.bash_profile` file located in the cpanel home directory.
4. **Create the custom cron XML configuration file if it is not created yet**

   We recommend to add all custom cron jobs into the custom cron configuration file, not to the standard cron XML configuration file.

   1) Create the custom cron configuration file, for example, `/hsphere/local/home/cpanel/shiva/custom/xml/cron_config.xml`

   2) The XML structure of the custom cron config file should be the same as of the standard cron configuration file.

   DTD scheme | Example

   3) Set the full pathname to the custom cron configuration file in hsphere.properties (`/hsphere/local/home/cpanel/shiva/psoft_config/hsphere.properties`):

   ```
   CRON_CONFIG = /hsphere/local/home/cpanel/shiva/custom/xml/cron_config.xml
   ```


5. Add the custom cron job to the custom cron configuration file

1) If the custom cron configuration is not created, create it according to the procedure specified in the previous step.

2) Add a new group of jobs if required or add your custom cron job there, or, add your custom job to an existing group. Group is set by the `group` tag. Jobs are set in the `cron` tags within their groups. For example:

```
<config>
...
  <group name="GroupN" maxpriority="10" defpriority="3">
  <!-- priority of jobs within the group is ranked from 1 to 10;
       if job priority is not specified, it is 3 by default -->
  ...
    <job name="CustomCron" class="custom.cron.CustomCron"
      starttime="5:00" period="1440"/>
    <!-- job starts every day (1440 minutes) at 5am -->
  ...
  <group>
</config>
```

XML syntax is described in the CP Cron Configuration guide.

Read *how to merge XML config files* in Customization Guide.

Related Docs: · CP Cron Configuration

Adding Custom Promotion Validators and Calculators

H-Sphere allows to define custom promotions - flexible discount systems - and assign them for individual plans.

Usually, a customer enters a certain promotion code on signup, the system verifies the entered code if it proves to be valid and corresponds to the chosen plan, that user signs up with that discount. This is called a codeable promotion. There are also codeless promotions that don't require a code. Discount depends on a particular promotion and the way the discount is calculated.

This document explains how to add and configure custom promotion validators and calculators.

Creating a Custom Promotion Validator

To add a custom promotion validator, use the the *psoft.hsphere.promotion.PromoValidator* Java class interface. The class can also extend the *AbstractPromoDataStorage* class to store the validator's data in the H-Sphere database.

Creating a Custom Promotion Calculator

To add a custom promotion validator, use the *psoft.hsphere.promotion.calc.PromoCalculator* Java class interface. The class can also extend the *AbstractPromoDataStorage* class to store the calculator's data in the H-Sphere database.

Example:

```
package psoft.hsphere.promotion.calc;

import psoft.hsphere.promotion.AbstractPromoDataStorage;
import psoft.hsphere.Session;
import psoft.hsphere.Account;
import psoft.util.USFormat;

import java.util.Hashtable;

public class PercentDiscountCalc
        extends AbstractPromoDataStorage implements PromoCalculator {
private double discountPercent;

    public PercentDiscountCalc(long promoId) throws Exception {
        super(promoId);
```



```
        discountPercent =
            USFormat.parseDouble((String) data.get("discount_percent"));
    }

    public PercentDiscountCalc(long promoId, Hashtable data) throws Exception {
        super(promoId, data);
        discountPercent =
            USFormat.parseDouble((String) data.get("discount_percent"));
    }

    public int getDataType() {
        return 2;
    }

    public double getPromoDiscount(Account a, double sum) {
        return sum*discountPercent/100;
    }

    public void updateData(long promoId, Hashtable data) throws Exception {
        super.updateData(promoId, data);
        Session.getLog().debug("Updating discount percent");
        discountPercent =
            USFormat.parseDouble((String) data.get("discount_percent"));
    }
    }
```

Configuration of Custom Promotion Validators and Calculators

All promotion validators and calculators should be added to and configured in the `promotions.xml` file. Its default location is `~cpanel/shiva/psoft/hsphere/promotion/xml/`. The file location can be altered with the `PROMO_CONFIG` and `CUSTOM_PROMO_CONFIG` properties in hsphere.properties.

PROMO_CONFIG=/hsphere/local/home/cpanel/shiva/psoft/hsphere/promotion/xml/promotions.xml
CUSTOM_PROMO_CONFIG=/hsphere/local/home/cpanel/shiva/custom/xml/promotions.xml

`promotions.xml` can be customized according to *XML customization rules* outlined in Customization Guide.

DTD structure | **Example**

Elements and attributes:

- <promotions> - contains all defined promotion validators.
- <promo> - configuration of a particular promotion validator.

  Attributes:

  - id - a unique promotion identifier: 1,2,...
  - description - a short description of the validator. It shows up in the dropdown list of promotions in Control Panel.
  - class - a name of a Java class for this validator.
  - itype - a type of the validator's interface. If it is set to "AUTO", interface for adding the validator parameter will be built automatically, depending on parameters defined for the validator or calculator; otherwise, values of the add_template and edit_template attributes will be used as template names for adding/editing set of data which is required for the given promo validator.
  - add_template - a name of a template for adding the validator data; the value of this attribute will take effect only in case if itype is not set to "AUTO".
  - edit_template - a name of a template for editing the validator data; the value of this attribute will take effect only in case if itype is not set to "AUTO".
- <calculators> - contains all defined promotion calculators.
- <calc> - configuration of a particular promotion calculator. Attributes have the same meaning as for promotion validators.
- <params> - contains all parameters used by promotion validators or calculators.
- <param> - configuration of a particular parameter.

  Attributes:

  - label - a name of a label in ~cpanel/psoft/hsphere/lang/hsphere_lang.properties with the message that appears on the page where users enters a promo code.
  - name - the parameter's name. The AbstractPromoDataStorage object stores validators' or calculators' data from HttpRequest using this name with the prefix 'pv_' for a validator and 'pc_' for a calculator. I.e., if name="percent", AbstractPromoDataStorage expects the 'pv_percent' variable for the validator and the 'pc_percent' variable for the calculator to be present in HttpRequest.

Merchant Gateway SDK

H-Sphere Merchant Gateway SDK is a development tool to integrate *merchant gateways* as *installable H-Sphere packages (.hsp)*. The SDK allows developing and testing gateway packages outside H-Sphere.

To use Merchant Gateway SDK, make sure you have the following software installed:

- Java 1.3 or higher
- Apache Ant 1.6.1 or higher - a Java-based build tool
- JUnit - a framework to write repeatable tests

Download Latest Merchant Gateway SDK | Online Documentation

Using EasyApp SDK

(version 2.4.3 and higher)

Related Docs: · Building Packages

EasyApp SDK is a standard development kit for easy integration of PHP/MySQL applications to H-Sphere. The goal of SDK is to provide the developer with the ability to generate H-Sphere package without programming any java or html UI code. It should be useful to integrate applications like phpBB2, Mambo, phpMyChat, b2evolution, PHPnuke, etc.

The user gets a web installer which collects application installation data (db username, db password, email, etc.), then writes it to config files and populates app database with the specified dump.

Positive Software offers its own collection of over 30 popular PHP/MySQL applications. They are available for download from our website.

Building Application Packages

Control Panel, Linux&FreeBSD part	Windows

1. Download and install:
  1. Java 1.3 or higher
  2. Apache Ant 1.6.1 or higher - a Java-based build tool.
  3. JUnit - a framework to write repeatable tests. It is recommended but optional.
2. Log in as the *cpanel user*.
3. Download EasyApp SDK zip.
4. Unzip generated SDK from the `dist` directory to the server (or under the user) where you will build the package.
	(H-Sphere Winbox 2.4.3 Patch 2 and up)
	
	1. Download the latest version of WiX and install it.
	2. Download NAnt 0.85RC3 or older and install it.
	3. Register paths to WiX and NAnt tools in the PATH environment variable.
	4. Register path to the currently used .NET Framework in the DOTNETDIR environment variable (for instance, `C:\WINNT\Microsoft.NET\Framework\v1.1.4322`).
	5. Download the `EasyApp_MSI_SDK_<hs_version_code>_<build>.zip` file from the EasyApp MSI SDK download directory, where

5. Create the `collection` directory. The contents of the original `collection` directory with only one application as an example is available for reference. You can also download the entire directory with all applications from our own collection (~50 MB).
6. Setup a separate directory inside the collections directory for each application that you want to include into the package.
7. Create main.xml descriptor in the application directory for each application you want in the package.
8. Place application tarball in the application directory (.tar.gz, .tgz, .zip extensions are supported)
9. Place application configs, sql templates, and logo into the application directory.
10. Inside the collections directory create collection.xml file.
11. Inside the collections directory, create `menu & hsphere_lang language` files with all the labels used.
12. Place icons for different h-sphere designs into the collections directory. The icons must represent your package on the home page of the user control panel.
13. Setup yafv file for the collection file.
14. Type
    ant
    As the result, you should get h-sphere package in the `pkg/` directory.

Now you can install the package according to our documentation on *package installation*.

`hs_version_code` is your H-Sphere Winbox version code (for example, 243p7, 250b4, 250rc1), and `build` is the latest available build number for this EasyApp MSI SDK version. Then, unzip the downloaded archive to any appropriate location.

6. Create `EasyApp` directory for EasyApp MSI SDK.
7. Create `easyapp-col` directory inside the `EasyApp` directory.
8. Setup a separate directory inside the `easyapp-col` directory for each application that you want to include into the package.
9. Create main.xml descriptor in the application directory for each application you want in the package.
10. Place application tarball in the application directory (.tar.gz, .tgz, .zip extensions are supported)
11. Place application configs, sql templates, and logo into the application directory.
12. Inside the `easyapp-col` directory create collection.xml file.
13. Inside the `easyapp-col` directory, create `menu & hsphere_lang language` files with all the labels used.
14. From the EasyApp MSI SDK directory type
    nant
    An MSI package should be created in this directory.

Now you can install the package according to our documentation on *EasyApp MSI installation*.

Related Docs: · Building Packages

EasyApp SDK descriptor

(version 2.4.3 and up)

Related Docs: · Using EasyApp SDK

For each application you want to package via EasyApp SDK, you need to create a descriptor. Inside application directory create main.xml. The file starts with app tag.

There are several tags inside:

name	the name of the project (alphanumeric), will be used as resource type in h-sphere
description	will be used as resource description
version	Application version
title	label to use as a title of the application
intro	label to use as basic decription/intro for the app
tarball	name of the tarball/bundle within the application. File under the same name should be in application directory
size	disk space that script requires, in MB
license	attribute type="XXX" - specifies the license type, value specifies URL to the license. If known type is used, no url is necessary. Known types: GPL", "BSD", "MPL"
homepage	url to application homepage
donation	url to donation page
documentation	url to documentation page
forum	url to support forum
logo	image name - logo image, should be present in the application directory
variables	list of configurational parameters/variables user will be prompted with during install/or when he wants to change parameters
install	list of actions to perform during the installation of application; it can contain optional `done` tag that describes actions performed at the end

	of the installation, if any. It can include other tags: • `iframe` - link to the original installation page the user will be taken to at some stage of the installation. You can use this tag if it's too complicated to perform the entire procedure in H-Sphere. • `access` - gives link to application or its parts the user will be taken to after the installation is finished, e.g. page of the application's administrator.
change	list of actions to perform on change
uninstall	list of actions to perform on uninstall
upgrade	attribute from="XXX" - where XXX is previously defined version. There can be multiple upgrade tags, from different versions. Inside upgrade tag, there is actions tag for actions to perform during upgrade. The tag is used to automate upgrades of the application
lang	language files bundled with package.

Sections

- Section <lang> - Specifies which lang file must be used for this application
- Sections <install>,<remove> - Describes the actions that constitute install/uninstall of the application software.
- Section <upgrade> - Used for possible autoupgrades, just like in <install>, <uninstall>, and <change>, but in the title requries the number of the version you upgrade from.
  Example:<upgrade from="2.9.7">

Here is an example of descriptor for Mambo server.

Related Docs: · Using EasyApp SDK

EasyApp SDK Collection descriptor

(version 2.4.3 and up)

Related Docs: · Using EasyApp SDK

collection.xml is located in the collections directory (typically easyapp-col/) and stores information about the applications included in the package.

id	the type id of the resource in H-Sphere, e.g. 20011. For H-Sphere packages, we use IDs starting with 20000, 20000 to 20010 are already taken. Third party developers must reserve IDs in http://www.psoft.net/my-hsphere/
name	the name of the resource, e.g. easyapp
description	will be used as resource description
version	package version
group	application category. The ID of the application is added to the custom template filename for each application, e.g. info-19.html, install-19.html etc.
lang	language files bundled with the package
icon	icons representing the package resource in different H-Sphere designs
menu	where in the menu the link will show

Related Docs: · Using EasyApp SDK

EasyApp SDK Variables

(version 2.4.3 and up)

Related Docs: · Using EasyApp SDK

The variables sections define variables used by applications. Each variable is defined by "field" type.

The following attributes can be defined for each field:

name	field name
type	field type, posible values: mysql, text, checkbox, textarea, select
label	the lable to show in html
default	default value
max	max size
change	if 1, the variable can be updated at later time
upgrade	if 1, during the package upgrade, this field will be added to existing tables
check	this attribute activates client-side form validation (YAFV) mechanism for this field

---

Related Docs: · Using EasyApp SDK

EasyApp SDK Actions

(version 2.4.3 and up)

Related Docs: · Using EasyApp SDK

EasyApp can perform actions to install/uninstall/upgrade software. There are several defined actions:

actions	parameters	description
uncompress	target - directory where to place uncompressed files	uncompresses default tarball, and moves file into target directory below webspace for selected domain
remove	target - directory to remove	removes files from the directory under the given domain name
sql	mysql - mysql var, value corresponds to target template	processed defined template, and uses it as SQL to import into specified db. SDK will automatically include template into the package
config	target - where to place config file, name - config file name, value = template to use	Processes template and places it under domain/target/name, automatically stores template in package
move_safe	from - target direcotry inside domain, to - directory in the safe zone	takes domain/place_where_app_installed/from directory and moves it under safe_zone/to directory outside of the web accessible space.
remove_safe	target - directory to remove, in the safe zone	removes directory in safe zone
config_safe	target - where to place config file, name - config file name, value = template to use	same as config, but in safe zone

safe_zone - each easyapp can have "safe_zone" directory, where it can store files outside of the web accessible space.

Related Docs: · Using EasyApp SDK

