CITRIX

CloudPlatform

(powered by Apache CloudStack)

User Interface Customization Guide

For CloudPlatform Version 4.2

Revised November 11, 2013 9:40 PM

Contents

Getting More Information and Help

Additional Documentation

You can find out more about CloudPlatform's many additional features and installation options in the other CloudPlatform documentation available at http://docs.citrix.com/cloudplatform.

- CloudPlatform Release Notes
- CloudPlatform Trial Installation Guide
- CloudPlatform Installation Guide
- CloudPlatform Administrator's Guide
- CloudPlatform Developer's Guide
- CloudPlatform API Reference

Citrix Knowledge Center

Troubleshooting articles by the Citrix support team are available in the Citrix Knowledge Center at http://support.citrix.com/product/cs/ .

Contacting Support

The support team is available to help customers plan and execute their installations. To contact the support team, log in to the support portal at http://support.citrix.com/cloudsupport with the account credentials you received when you purchased your support contract.

Introduction

The CloudPlatform™ User Interface (UI) is a rich AJAX client interface that allows you to manage all aspect of the cloud and is a complete reference implementation of the CloudPlatform API. The CloudPlatform UI supports three access roles.

- **Root Admin**. Access to all features of the cloud, including both virtual and physical resource management.
- **Domain-Admin**. Access to only the virtual resources of the clouds that belong to the administrator's domain.
- **User**. Access to only the features that allow management of the user's virtual instances, storage, and network.

This document describes the various methods of user interface customization from simple branding to a complete redesign.

License

Supportability of Modified UI

Per our license, you are welcome to modify, add, or reuse any part of the CloudPlatform UI to suit your needs. However, once modified, Citrix can no longer support any defects resulting from the customization nor can we support any upgrade process to future releases of the CloudPlatform Management Server.

Customization

The CloudPlatform UI is built entirely on HTML/JSP, CSS, Javascript, and uses jQuery 1.4 as the Javascript Library for all AJAX calls, event handling, and animations. You can find the latest jQuery reference API at http://api.jquery.com/. We recommend that any changes be made by someone with development experience in the listed above technologies. We also recommend using a web development tool such as Firebug for Firefox to help inspect the various elements of the UI for easier modification.

Getting Started

To get started, log into your CloudPlatform Management Server and go to the following directory to find all the files and resources that make up the user interface.

/usr/share/cloud/management/webapps/client/

The following table will describe all the major files located in the directory used to build the user interface.

File	Description
index.jsp	The main HTML/JSP page. This is where all CSS and Javascripts are specified.
favicon.ico	Default favicon. Replace this to suit your needs.
dictionary.jsp	Localization strings for JavaScript
/images/*.*	Folder that contains all the images used by the CloudPlatform UI.
/css/cloudstack3.css	Main CSS file that contains most of the CSS definitions used by the CloudPlatform UI.
/plugins/*.*	Directory that includes custom HTML, CSS, and Javascript for the UI.
/scripts/jquery*.js	Javascript libraries used by the CloudPlatform UI. You should not have to modify any of these files.
/scripts/cloud.core.callback.js	Javascript file that you can modify if you wish to integrate the CloudPlatform UI as a single sign-on solution with your existing website/portal.
/scripts/cloud.core.js	Javascript file that contains the common functions used by the CloudPlatform UI.

/scripts/cloud.core.init.js	Javascript file that contains the default initialization logic for the CloudPlatform UI. This is also the location where you need to specify the default API URL for AJAX calls if you decide to change the default URL.
/scripts/cloud.core.*.js	Javascript files that correspond to each major element presented in the CloudPlatform UI.

Simple Branding

Simple branding is defined as the replacement of the header logo and all major color schemes in the CloudPlatform UI to match your company's logo and colors. This includes modification of the color of the header, tabs, grid header, and all dialogs. To make these changes, use the reference CSS files found in /modules/cloudPlatform/cloudPlatform.css. You can also replace the /favicon.ico, /modules/cloudPlatform/logo.pngand any other images in /modules/cloudPlatform to replace the default images.

The illustration below identifies some of the major CSS definitions which are typically modified in simple branding.

Changing the API URL

The default host URL on a new installation of the CloudPlatform Management Service is http://<server>:<port>/client. Please refer to http://tomcat.apache.org/tomcat-5.5-doc/index.html for documentation on how to change default host URL.

An API URL may need to be changed for the following reasons.

- The default configuration of the Tomcat engine has been changed to your desired URL.
- You have a load balancer or proxy server that is fronting a public host URL that is different from what is currently configured as the default API URL.

If at any point the public API URL is different than what is configured by default, you will need to reconfigure your setup by changing the API path and modifying the cloud.core.init.js file. Use the following sections to make these changes.

To Edit the API Path

By default, the API URL as configured in the user interface is “client/api” which is relative to the default host DNS/IP. If you would like to change API path, use the following steps.

1. Edit the following file.

```
/usr/share/cloud/management/webapps/client/WEB-INF/web.xml
```

2. Within the file, find the following XML tag:

```
<servlet-mapping>
        <servlet-name>apiServlet</servlet-name>
        <url-pattern>/api/*</url-pattern>
</servlet-mapping>
```

3. Change the <url-pattern> tag to the desired API URL.

Once you have changed the API path, proceed to the next section for steps to modify the cloud.core.init.js file.

To Modify the cloud.core.init.js File

If the default API URL has changed from “client/api”, use the following steps to modify the cloud.core.init.js file.

1. Edit the following file.

```
/usr/share/cloud/management/webapps/client/scripts/cloud.core.init.js
```

2. Find the following jQuery definition in the file:

```
$.ajaxSetup({
      url: "/client/api",
      dataType: "json",
      cache: false,
      error: function(XMLHttpResponse) {
            handleError(XMLHttpResponse);
      },
      beforeSend: function(XMLHttpRequest) {
            if (g_mySession == $.cookie("JSESSIONID")) {
                  return true;
            } else {
                  $("#dialog_session_expired").dialog("open");
                  return false;
            }
      }
});
```

3. Modify the URL option (this option is highlighted in red in the above example) to your desired URL.

Once this has been modified, all subsequent AJAX calls will be made to the new URL. You may need to refresh the browser to update any cached Javascript files for the new settings to take place.

Localization

The process of localizing the CloudPlatform User Interface requires a two step process:

1. Adding a new properties file for the language you wish to localize your UI to. The file must be copied over to the following directory in your management server: /usr/share/cloud/management/webapps/client/WEB-INF/classes/resources. The default installation of CloudPlatform contains sample property files for simplified Chinese, Japanese, and Spanish. Simply make a copy of messages.properties and rename the new file using the naming convention of messages_<lang_code>_<country_code>.properties. Edit the file and translate the English text to your language of choice.

   The valid language codes can be found at http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt.

   The valid country codes can be found at http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html.

2. The second thing that is required is to make sure that you set the cookie, “lang” to be your language + country code. You can either modify the current UI to accommodate the new language or just ensure that you set the cookie yourself. To modify the UI, you would need to edit index.jsp, find the language drop down menu by searching for “lang_menu”, and finally adding a new <li> item to match your new language.

Example:

For example, if you wish to add a French localization, you would make a copy of messages.properties and rename it to messages_fr.properties. Edit the new file and translate the English text. You would then edit index.jsp, and add <li id="fr"><fmt:message key="label.lang.french"/> to the dropdown menu.

Changing Session Timeout

The default session timeout for the User Interface is 30 minutes as configured within Tomcat. If you wish to increase this timeout period, use the following steps.

1. Edit the following file.

```
/usr/share/cloud/management/webapps/client/WEB-INF/web.xml
```

2. Within the file, add the following XML tag or modify if it’s already there to the desired timeout period in minutes:

```
<session-config>
         <session-timeout>60</session-timeout>
</ session-config>
```

3. Reboot the Tomcat container:

```
# service cloud-management restart
```


Single Sign-on Integration

The user interface is created entirely using the session-based CloudPlatform API. Once a user account has successfully logged in, a JSESSIONID cookie is sent back as part of the authorization process that can be used until the session has expired on the server. As a result, there are multiple ways that single sign-on can be integrated. Two of these methods are discussed in detail in the following sections.

The two methods of integrating your portal to CloudPlatform depend on the modification of the "cloud.core.callbacks.js." This file includes a method, onLogoutCallback(), that can be implemented to redirect the user to your portal if the session times out. The other half of this file include a sample AJAX login API call to the CloudPlatform Management Server. You must make the login API from the CloudPlatform domain or the browser will reject any cross-browser script calls for security reasons. If your CloudPlatform Management Server and Portal exist within the same domain, you do not have to worry about this. Simply make the login API call from anywhere.

Traditional

The traditional way of integrating an existing portal with CloudPlatform is to execute the API command "login" on behalf of the user. Using this method, you would need to construct the login command and pass in the required parameters such as the username, account, domain, and password. Upon a successful response, you would only need to ensure that the global variable "g_loginResponse" is set to the JSON response of the login API call. A typical client side single sign-on implementation would look like the following:

- Portal has a link (or iframe) to the CloudPlatform interface. That link should contain enough information to contruct a proper login API call.
- A modified "cloud.core.callbacks.js" intercepts the referred link, constructs the "login" call, and executes it against /client/api URL.
- Upon a successful response, the JESSIONID cookie will be automatically set by the browser, and the global variable "g_loginResponse" should be set to the JSON response.

Shared Key

The shared key method is very similar to the traditional way except for the additional security by hashing the URL with a shared secret key when making the same login API command. The actual process of signing is very similar to the process described at http://download.cloud.com/releases/2.2/api/html/global_admin/2.2api_security_details.html under the "API/Secret key security section" with the following exceptions:

- You do not need to pass in the API Key
- The four parameters that must be passed in for the login command are domainId, username, timestamp, and signature.

A sample login request:

```
https://<server>:8080/client/api?command=login&username=XXX&domainid=NNN&time
stamp=YYY&signature=<secure-hash>
```


To get the single sign-on secret key, you must retrieve it from the CloudPlatform database under the configuration table for the key “security.singlesignon.key”. Copy this key to the application you wish to integrate CloudPlatform with and follow the above instructions to sign the login command.

The timestamp parameter is simply the current system time in milliseconds. There is also a fault tolerance configurable value in the configuration table, “security.singlesignon.tolerance.millis” that can be changed to suit your preference. The timestamp passed in as part of the login request needs to be within the CloudPlatform Management Server time plus the fault tolerance time.

Cross Site Request Forgery (CSRF)

The CloudPlatform Management User Interface protects itself from CSRF attacks. Additional information about this can be found at http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).

To protect the User Interface from CSRF attacks, a sessionkey response value is returned upon a successful login attempt. This sessionkey is then passed with all subsequent API command calls. This is different from the JESSSIONID and should never be stored in a cookie.

If you plan on implementing your own User Interface on top of the CloudPlatform Query API, you must ensure the following when using the sessionkey:

- Should *__not__* be stored as a cookie
- Must be returned with every request, for example:

```
http://<server>:8080/client/api?command=XXX&sessionKey=YYY
```

If you send any subsequent requests without a valid sessionkey, a 401 Unauthorized HTTP error code will be returned.

Third-Party UI Plugin Framework

Using the third-party plugin framework, you can write and install extensions to CloudPlatform. Once installed and enabled, your plugins will appear in the UI alongside the other CloudPlatform features. The code for the plugin is simply placed in a special directory within CloudPlatform's installed code at any time after installation. The new plugin appears only when it is enabled by the cloud administrator.

The left navigation bar of the UI has a new Plugins button to help you work with UI plugins.

How to Write a Plugin: Overview

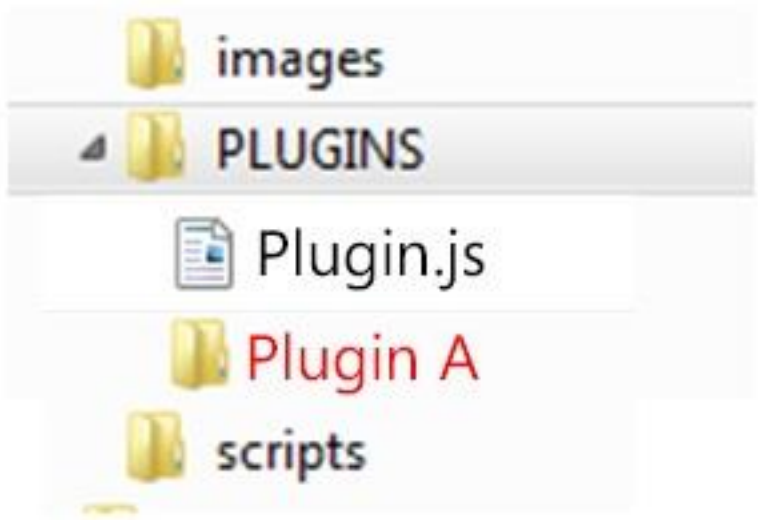
The basic procedure for writing a plugin is:

1. Write the code and create the other files needed. You will need the plugin code itself (in Javascript), a thumbnail image, the plugin listing, and a CSS file.

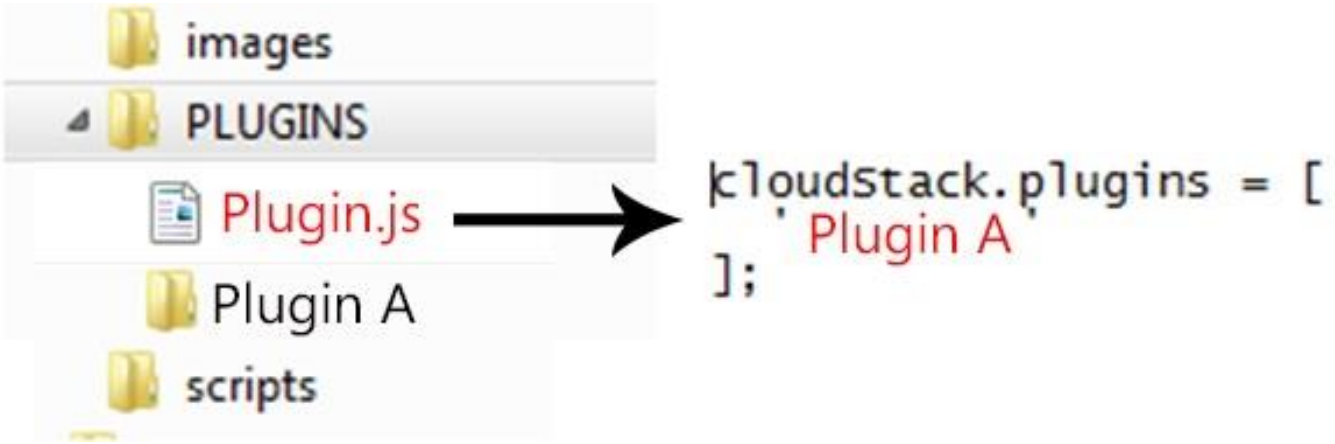
All UI plugins have the following set of files:

```
+-- cloudstack/
  +-- ui/
    +-- plugins/
      +-- csMyFirstPlugin/
        +-- config.js --> Plugin metadata (title, author, vendor URL, etc.)
        +-- icon.png --> Icon, shown on side nav bar and plugin listing
                         (should be square, and ~50x50px)
        +-- csMyFirstPlugin.css --> CSS file, loaded automatically when
                                    plugin loads
        +-- csMyFirstPlugin.js --> Main JS file, containing plugin code
```

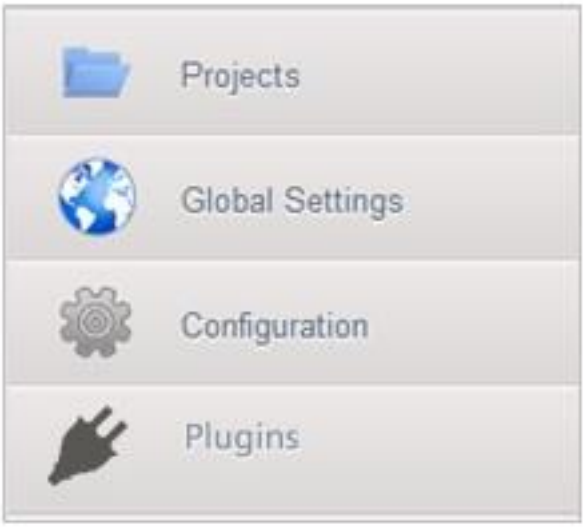

2. The CloudPlatform administrator adds the folder containing your plugin code under the PLUGINS folder.



3. The administrator also adds the name of your plugin to the plugin.js file in the PLUGINS folder.



4. The next time the user refreshes the UI in the browser, your plugin will appear in the left navigation bar.



How to Write a Plugin: Implementation Details

This section requires an understanding of JavaScript and the CloudPlatform API. You don't need knowledge of specific frameworks for this tutorial (jQuery, etc.), since the UI handles the front-end rendering for you.

There is much more to the CloudPlatform UI framework than can be described here. The UI is very flexible to handle many use cases, so there are countless options and variations. The best reference right now is to read the existing code for the main UI, which is in the /ui folder. Plugins are written in a very similar way to the main UI.

1. Create the directory to hold your plugin.

   All plugins are composed of set of required files in the directory /ui/plugins/pluginID, where pluginID is a short name for your plugin. It's recommended that you prefix your folder name (for example, csMyPlugin) to avoid naming conflicts with other people's plugins. In this example, the plugin is named csMyFirstPlugin.

```
$ cd cloudstack/ui/plugins
$ mkdir csMyFirstPlugin
$ ls -l
total 8
drwxr-xr-x 2 bgregory staff 68 Feb 11 14:44 csMyFirstPlugin
-rw-r--r-- 1 bgregory staff 101 Feb 11 14:26 plugins.js
```

2. Change to your new plugin directory.

```
$ cd csMyFirstPlugin
```

3. Set up the listing.

   Add the file config.js, using your favorite editor.

```
$ vi config.js
```

   Add the following content to config.js. This information will be displayed on the plugin listing page in the UI.

```
(function (cloudStack) {
```



```
  cloudStack.plugins.csMyFirstPlugin.config = {
    title: 'My first plugin',
    desc: 'Tutorial plugin',
    externalLink: 'http://www.cloudstack.org/',
    authorName: 'Test Plugin Developer',
    authorEmail: 'plugin.developer@example.com'
  };
}(cloudStack));
```

4. Add a new main section.

   Add the file csMyFirstPlugin.js, using your favorite editor.

```
$ vi csMyFirstPlugin.js
```

   Add the following content to csMyFirstPlugin.js:

```
function (cloudStack) {
  cloudStack.plugins.csMyFirstPlugin = function(plugin) {
    plugin.ui.addSection({
      id: 'csMyFirstPlugin',
      title: 'My Plugin',
      preFilter: function(args) {
        return isAdmin();
      },
      show: function() {
        return $('<div>').html('Content will go here');
      }
    });
  };
}(cloudStack));
```

5. Register the plugin.

   You now have the minimal content needed to run the plugin, so you can activate the plugin in the UI by adding it to plugins.js. First, edit the file:

```
$ cd cloudstack/ui/plugins
$ vi plugins.js
```

   Now add the following to plugins.js:

```
(function($, cloudStack) {
  cloudStack.plugins = [
    'csMyFirstPlugin'
  ];
}(jQuery, cloudStack));
```

6. Check the plugin in the UI.

   First, copy all the plugin code that you have created so far to /tomcat/webapps/client/plugins. Then refresh the browser and click Plugins in the side navigation bar. You should see your new plugin.

7. Make the plugin do something.

   Right now, you just have placeholder content in the new plugin. It's time to add real code. In this example, you will write a basic list view, which renders data from an API call. You will list all virtual machines owned by the logged-in user. To do this, replace the 'show' function in the plugin code with a 'listView' block, containing the required syntax for a list view. To get the data, use the listVirtualMachines API call. Without any parameters, it will return VMs only for your active user. Use the provided 'apiCall' helper method to handle the server call. Of course, you are free to use any other method for making the AJAX call (for example, jQuery's $.ajax method).

   First, open your plugin's JavaScript source file in your favorite editor:

```
$ cd csMyFirstPlugin
$ vi csMyFirstPlugin.js
```

   Add the following code in csMyFirstPlugin.js:

```
(function (cloudStack) {
  cloudStack.plugins.csMyFirstPlugin = function(plugin) {
    plugin.ui.addSection({
      id: 'csMyFirstPlugin',
      title: 'My Plugin',
      preFilter: function(args) {
        return isAdmin();
      },

      // Render page as a list view
      listView: {
        id: 'testPluginInstances',
        fields: {
          name: { label: 'label.name' },
          instancename: { label: 'label.internal.name' },
          displayname: { label: 'label.display.name' },
          zonename: { label: 'label.zone.name' }
        },
        dataProvider: function(args) {
          // API calls go here, to retrive the data asynchronously
          //
          // On successful retrieval, call
          // args.response.success({ data: [data array] });
          plugin.ui.apiCall('listVirtualMachines', {
            success: function(json) {
              var vms = json.listvirtualmachinesresponse.virtualmachine;
              args.response.success({ data: vms });
            },
            error: function(errorMessage) {
              args.response.error(errorMessage)
            }
          });
        }
      }
    });
```



```
  };
}(cloudStack));
```

8. Test the plugin.

   First, copy all the plugin code that you have created so far to /tomcat/webapps/client/plugins. Then refresh the browser. You can see that your placeholder content was replaced with a list table, containing 4 columns of virtual machine data.

9. Add an action button.

   Let's add an action button to the list view, which will reboot the VM. To do this, add an actions block under listView. After specifying the correct format, the actions will appear automatically to the right of each row of data.

```
$ vi csMyFirstPlugin.js
```

   Now add the following new code in csMyFirstPlugin.js. (The dots ... show where we have omitted some existing code for the sake of space. Don't actually cut and paste that part):

```
...
  listView: {
    id: 'testPluginInstances',
    ...

    actions: {
      // The key/ID you specify here will determine what icon is
      // shown in the UI for this action,
      // and will be added as a CSS class to the action's element
      // (i.e., '.action.restart')
      //
      // -- here, 'restart' is a predefined name in that will
      // automatically show a 'reboot' arrow as an icon;
      // this can be changed in csMyFirstPlugin.css
      restart: {
        label: 'Restart VM',
        messages: {
          confirm: function() {
            return 'Are you sure you want to restart this VM?'
          },
          notification: function() { return 'Rebooted VM' }
        },
        action: function(args) {
          // Get the instance object of the selected row from context
          //
          // -- all currently loaded state is stored in 'context' as objects,
          // such as the selected list view row,
          // the selected section, and active user
          //
          // -- for list view actions, the object's key will be the same as
          // listView.id, specified above;
          // always make sure you specify an 'id' for the listView,
```



```
          // or else it will be 'undefined!'
          var instance = args.context.testPluginInstances[0];
          plugin.ui.apiCall('rebootVirtualMachine', {
            // These will be appended to the API request
            //
            // i.e., rebootVirtualMachine&id=...
            data: {
              id: instance.id
            },
            success: function(json) {
              args.response.success({
                // This is an async job, so success here only indicates
                // that the job was initiated.
                //
                // To pass the job ID to the notification UI
                // (for checking to see when action is completed),
                // '_custom: { jobID: ... }' needs to always be passed on
                // success, in the same format as below
                _custom: { jobId: json.rebootvirtualmachineresponse.jobid }
              });
            },
            error: function(errorMessage) {
              args.response.error(errorMessage);
              // Cancel action, show error message returned
            }
          });
        },
        // Because rebootVirtualMachine is an async job, we need to add
        // a poll function, which will perodically check
        // the management server to see if the job is ready
        // (via pollAsyncJobResult API call)
        //
        // The plugin API provides a helper function,
        // 'plugin.ui.pollAsyncJob',
        // which will work for most jobs in CloudPlatform
        notification: {
          poll: plugin.ui.pollAsyncJob
        }
      }
    },

    dataProvider: function(args) {
      ...
  ...
```

10. Add the thumbnail icon.

    Create an icon file; it should be square, about 50x50 pixels, and named icon.png. Copy it into the same directory with your plugin code: cloudstack/ui/plugins/csMyFirstPlugin/icon.png.

11. Add the stylesheet.

Create a CSS file, with the same name as your .js file. Copy it into the same directory with your plugin code: cloudstack/ui/plugins/csMyFirstPlugin/csMyFirstPlugin.css.

