Citrix XenCenter Plugin Specification

September 2014

Version 1.1.0

Citrix XenCenter Plugin Specification Guide

Citrix, Inc.
851 West Cypress Creek Road
Fort Lauderdale, FL 33309
United States of America

Contents

Introduction

This document explains how to write a plugin to Citrix XenCenter, the graphical user interface for XenServer. Using the plugin mechanism third-parties can:

- Create new menu entries in the XenCenter menus linked to an executable file or PowerShell script, including full use of the XenServerPSSnapIn PowerShell extensions.
- Cause a URL to be loaded into a tab in XenCenter.

The XenCenter plugin mechanism is context aware, allowing you to use XenSearch to specify complicated queries. Additionally plugins can take advantage of contextual information passed as arguments to executables or as replaceable parameters in URLs.

A XenCenter plugin consists of the following components:

- An XML configuration file
- A resource DLL for each supported locale. Currently XenCenter exists in English and Japanese versions only.
- The application and any resources it requires

Put these components of a plugin in a subdirectory of the XenCenter installation directory. For example, a default installation of XenCenter requires that a plugin reside in:

```
C:\Program Files\Citrix\XenCenter\Plugins\<organization_name>\<plugin_name>
```

XenCenter loads all valid plugins found in subdirectories of the `Plugins` directory when it starts:

- The plugin name (`<plugin_name>`) must be the same as the directory in which it is placed.
- The resource DLL and the XML configuration file must follow these naming conventions:

```
<plugin_name>.resources.dll
<plugin_name>.xcplugin.xml
```

For example, if your organization is called Citrix and you write a plugin called Example which runs a batch file called do_something.bat, the following files must be in place (assuming a default XenCenter installation):

```
C:\Program Files\Citrix\XenCenter\Plugins\Citrix\Example\Example.resources.dll
C:\Program Files\Citrix\XenCenter\Plugins\Citrix\Example\example.xcplugin.xml
C:\Program Files\Citrix\XenCenter\Plugins\Citrix\Example\do_something.bat
```

XML Configuration File

In your plugin XML configuration file you define which menu items and tab items you would like to appear in XenCenter. These are referred to as *features*, and you can customize each one using the XML attributes described in this specification.

Example: A sample XML configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XenCenterPlugin PUBLIC "-//XENCENTERPLUGIN//DTD XENCENTERPLUGIN1//EN" "xencenter-1.dtd">
<XenCenterPlugin
        xmlns="http://www.citrix.com/XenCenter/Plugins/schema"
        version="1"
        plugin_version="1.0.0.0">
        <MenuItem
                name="Hello World"
                menu="vm"
                contextmenu="none"
                serialized="obj"
                icon="Plugins\Citrix\HelloWorld\icon.png"
                search="dd7fbce2-b0d4-4c61-9707-e4b0f718673e"
                description="The worlds most friendly plugin, it loves to say hello">
                <Shell
                        filename="Plugins\Citrix\HelloWorld\HelloWorld.exe"
                />
        </MenuItem>
        <TabPage
                name="Google"
                url="http://www.google.com/"
        />
        <Search
                uuid="dd7fbce2-b0d4-4c61-9707-e4b0f718673e"
                name="HelloSearch"
                major_version="2"
                minor_version="0"
                show_expanded="yes">
                <Query>
                        <QueryScope>
                                <VM />
                        </QueryScope>
                        <RecursiveXMOListPropertyQuery
                                property="vm">
                                <EnumPropertyQuery
                                        property="power_state"
                                        equals="no"
                                        query="Running" />
                        </RecursiveXMOListPropertyQuery>
                </Query>
        </Search>
```

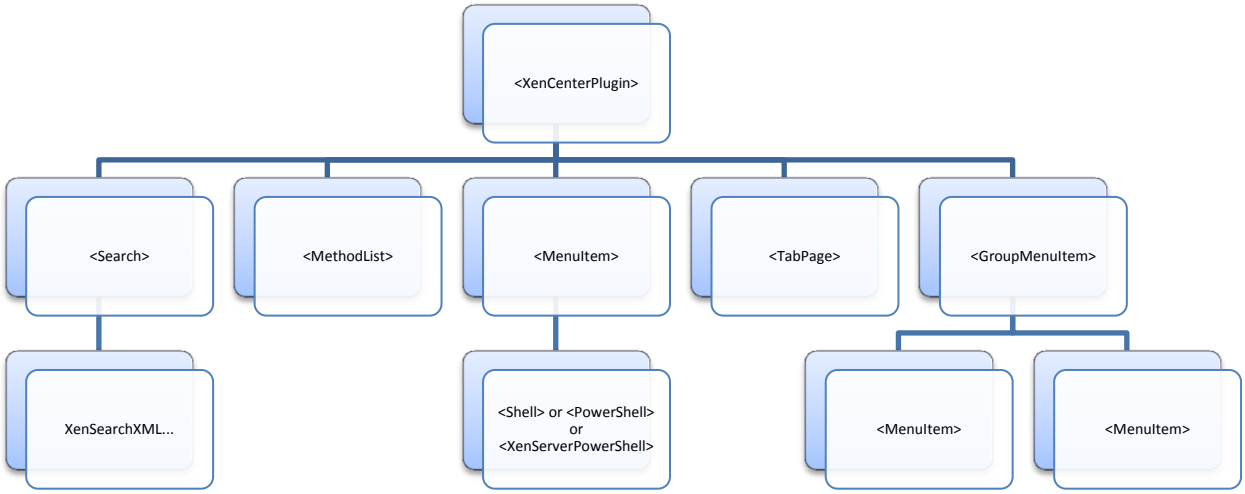
```
</XenCenterPlugin>
```

If your configuration file is invalid, XenCenter logs an error and the plugin is ignored. You can enable, disable and view errors with your plugin configuration file in the XenCenter plugins dialog.

Note: Errors will only be shown in the dialog your configuration file XML can be parsed. If your plugin is not listed in the dialog use the XenCenter log file to debug your XML.

XML Configuration File: Basic Structure

The XML elements in a configuration file have the following structure:



The `XenCenterPlugin` XML element is declared as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XenCenterPlugin PUBLIC "-//XENCENTERPLUGIN//DTD XENCENTERPLUGIN1//EN" "xencenter-1.dtd">
<XenCenterPlugin
        xmlns="http://www.citrix.com/XenCenter/Plugins/schema"
        version="1">
...
</XenCenterPlugin>
```

With the following XML attributes:

Important: The 'version' attribute is required; your plugin will not load unless it is set.

Key	Value	Description	Optional/ Required	Default
version	• 1	The XenCenter plugin version this plugin was written for. Currently only version 1 is in use.	Required	-
label	• [string]	Used in XenCenter as a user friendly replacement for the plugin name as dictated by the directory structure.	Optional	-
description	• [string]	A short description of this plugin to show in XenCenter.	Optional	-
copyright	• [string]	A copyright notice for this plugin to show in XenCenter	Optional	-
link	• [string]	A URL web resource for the plugin to show in XenCenter	Optional	-

XML Configuration File: XenSearch

You can include XenSearch definitions in your plugin configuration file for features to reference. They can use these searches to tell XenCenter when they wish to be enabled or shown.

Example: A MenuItem feature is using a XenSearch definition to restrict itself to shut down VMs

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XenCenterPlugin PUBLIC "-//XENCENTERPLUGIN//DTD XENCENTERPLUGIN1//EN" "xencenter-1.dtd">
<XenCenterPlugin
        xmlns="http://www.citrix.com/XenCenter/Plugins/schema"
        version="1">
        <MenuItem
                name="hello-menu-item"
                menu="vm"
                serialized="none"
                search="dd7fbce2-b0d4-4c61-9707-e4b0f718673e">
                <Shell
                        filename="Plugins\Citrix\HelloWorld\HelloWorld.bat"
                />
        </MenuItem>
        <Search
                uuid="dd7fbce2-b0d4-4c61-9707-e4b0f718673e"
                name="NotRunning"
                major_version="2"
                minor_version="0"
                show_expanded="yes">
                <Query>
                        <QueryScope>
```

```
                                <VM />
                        </QueryScope>
                        <RecursiveXMOListPropertyQuery
                                property="vm">
                                <EnumPropertyQuery
                                        property="power_state"
                                        equals="no"
                                        query="Running"
                                />
                        </RecursiveXMOListPropertyQuery>
                </Query>
        </Search>
</XenCenterPlugin>
```

The best way to get a XenSearch definition into your plugin configuration file is to construct it in XenCenter, export it to a local file and copy it into your configuration file.

XML Configuration File: Shared RBAC Method List

Important: Shared method list definitions were first added in version 5.6 Feature Pack 1.

To configure RBAC support for your plugin in version 5.6 you will need to define the list of required methods on each feature command individually. See 'Commands and RBAC' for more information.

When the user is connected to a server running Role Based Access Control your plugin may not have permission to execute all the XenServer API calls you need. A method list can be defined for each command to warn XenCenter which API calls will be needed before the plugin is even executed. See 'Commands and RBAC' for more information.

By defining a shared RBAC method list as a child of your XenCenterPlugin node you can have multiple commands share the same common list of methods:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XenCenterPlugin PUBLIC "-//XENCENTERPLUGIN//DTD XENCENTERPLUGIN1//EN" "xencenter-1.dtd">
<XenCenterPlugin
        xmlns="http://www.citrix.com/XenCenter/Plugins/schema"
        version="1">
        <MenuItem
                name="hello-menu-item"
                menu="vm"
                serialized="none">
                <Shell
                        filename="Plugins\Citrix\HelloWorld\HelloWorld.bat"
                        required_method_list="methodList1"
                />
        </MenuItem>
        <MenuItem
                name="hello-menu-item"
```

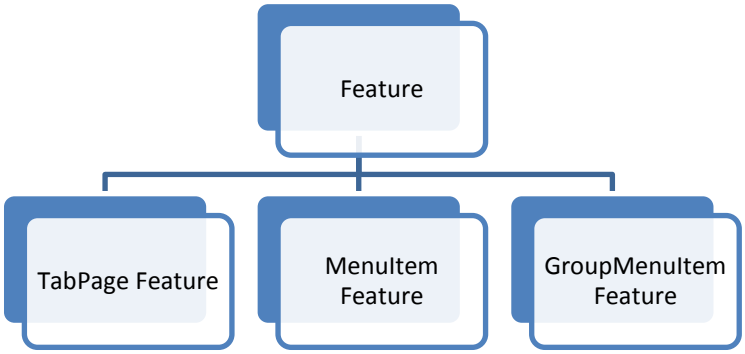
```
                menu="file"
                serialized="none">
                <Shell
                        filename="Plugins\Citrix\HelloWorld\HelloWorld.bat"
                        required_method_list=”methodList1”
                />
        </MenuItem>
        <MethodList name="methodList1">
                host.reboot, vm.start
        </MethodList>
</XenCenterPlugin>
```

This is purely for convenience and is equivalent to defining the method list on each command separately. When both `required_method_list` and `required_methods` are set on a command, `required_methods` is taken by preference.

For syntax and further information see ‘Commands and RBAC’.

Features

Each XenCenter Plugin can define multiple features to extend the functionality of XenCenter for specific tasks. These features are the new menu items and tab pages you are adding to XenCenter.



GroupMenuItem features are available to help organize your MenuItem features, but rely on MenuItem features to provide any functionality.

Features: XML Attributes

All of the features mentioned above share some common optional and required attributes that allow you to customize their appearance and functionality.

Important: The 'name' attribute is required for all features, your plugin will not load unless it is set

Key	Value	Description	Optional/ Required	Default
name	• [string]	The name for this feature. If label is not set, this name will be used for display purposes in XenCenter. It is also used for logging.	Required	-
label	• [string]	Used as a 'name' replacement for user facing display purposes in XenCenter.	Optional	-
search	• [string]	The uuid of a XenSearch defined in your configuration file. It will be used for setting enablement and visibility of this feature.	Optional	-
description	• [string]	A short description of this feature.	Optional	-
tooltip	• [string]	Any text you wish to display a tooltip for this feature.	Optional	-
icon	• [string]	A relative path from your XenCenter install directory to an	Optional	-

		icon image for this feature. It will be displayed at size 16x16.		

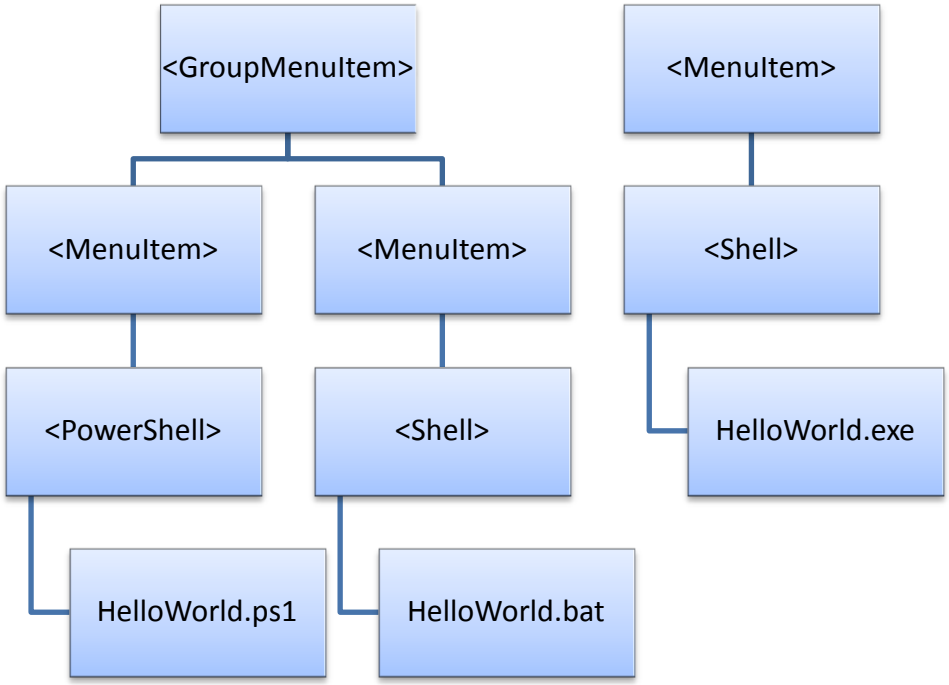
Example: A configuration file with a MenuItem feature using some of the common feature XML attributes

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XenCenterPlugin PUBLIC "-//XENCENTERPLUGIN//DTD XENCENTERPLUGIN1//EN" "xencenter-1.dtd">
<XenCenterPlugin
        xmlns="http://www.citrix.com/XenCenter/Plugins/schema"
        version="1">
        <MenuItem
                name="Hello Exe World"
                menu="file"
                serialized="obj"
                icon="Plugins\Citrix\HelloWorld\icon.png"
                tooltip="Says hello to the whole world"
                label="Hello"
                search="dd7fbce2-b0d4-4c61-9707-e4b0f718673e"
                description="The worlds most friendly plugin, it loves to say hello">
                <Shell
                        filename="Plugins\Citrix\HelloWorld\HelloWorld.exe"
                />
        </MenuItem>
        <Search
                uuid="dd7fbce2-b0d4-4c61-9707-e4b0f718673e"
                name="HelloSearch"
                major_version="2"
                minor_version="0"
                show_expanded="yes">
       <Query>
           <QueryScope>
               <VM />
           </QueryScope>
           <RecursiveXMOListPropertyQuery
                property="vm">
                <EnumPropertyQuery
                        property="power_state"
                        equals="no"
                        query="Running"
                />
           </RecursiveXMOListPropertyQuery>
       </Query>
   </Search>
</XenCenterPlugin>
```

Features: MenuItem and GroupMenuItem

Plugin authors can use MenuItem and GroupMenuItem features to add menu items in XenCenter. GroupMenuItems collect your menu items under sub menus and MenuItems launch your plugin commands.

Figure: An example hierarchy of menu items that launch various plugin commands



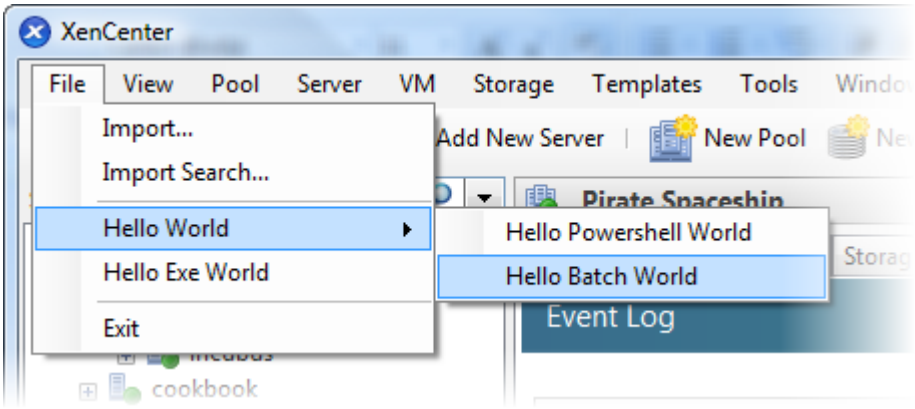
- Each GroupMenuItem can have multiple MenuItem children
- Each MenuItem has exactly one child command which runs a target executable or script.

Example: A configuration file detailing the MenuItems and GroupMenuItems shown in the figure above

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XenCenterPlugin PUBLIC "-//XENCENTERPLUGIN//DTD XENCENTERPLUGIN1//EN" "xencenter-1.dtd">
<XenCenterPlugin
        xmlns="http://www.citrix.com/XenCenter/Plugins/schema"
        version="1">
        <GroupMenuItem
                name="Hello World"
                menu="file">
                <MenuItem
                        name="Hello Powershell World"
                        menu="file"
                        serialized="obj">
                        <PowerShell
```

```
                                filename="Plugins\Citrix\HelloWorld\HelloWorld.ps1"
                        />
                </MenuItem>
                <MenuItem
                        name="Hello Batch World"
                        menu="file"
                        serialized="obj">
                        <Shell
                                filename="Plugins\Citrix\HelloWorld\HelloWorld.bat"
                        />
                </MenuItem>
        </GroupMenuItem>
                <MenuItem
                        name="Hello Exe World"
                        menu="file"
                        serialized="obj">
                        <Shell
                                filename="Plugins\Citrix\HelloWorld\HelloWorld.exe"
                        />
                </MenuItem>
</XenCenterPlugin>
```

MenuItems which are children of a GroupMenuItem appear as a sub menu under their group. The MenuItem validation logic still requires that the 'menu' attribute is set for these sub MenuItems, however their location is dictated by the menu attribute on the parent GroupMenuItem.



Features: MenuItem XML Attributes

Important: The inherited feature attribute 'name' is required and your plugin will not load unless it is set

Important: Each MenuItem feature must contain **exactly one** child node describing a XenCenter plugin command, your plugin will not load unless this is satisfied

Key	Value	Description	Optional/ Required	Default
-	• -	*[All attributes inherited from feature]*	-	-
menu	• file • view • pool • server • vm • storage • templates • tools • help	The XenCenter menu you would like this to appear under.	Required	-
serialized	• obj • global	If set to obj, the menu item will disable itself if it's command is already running against the selected object. If set to global only one instance of it's command will be allowed to run at a time regardless of what is selected.	Optional	-
contextmenu	• none • pool • server • vm • storage • template • folder	An additional context menu you would like this menu item to appear under. Unless you set 'none' the item will already be present on the context menu that relates to the 'menu' attribute (if such a context menu exists).	Optional	[value for menu]

Features: GroupMenuItem XML Attributes

Important: The inherited feature attribute 'name' is required and your plugin will not load unless it is set

Note: Each GroupMenuItem feature can contain as many MenuItem child nodes as you would like

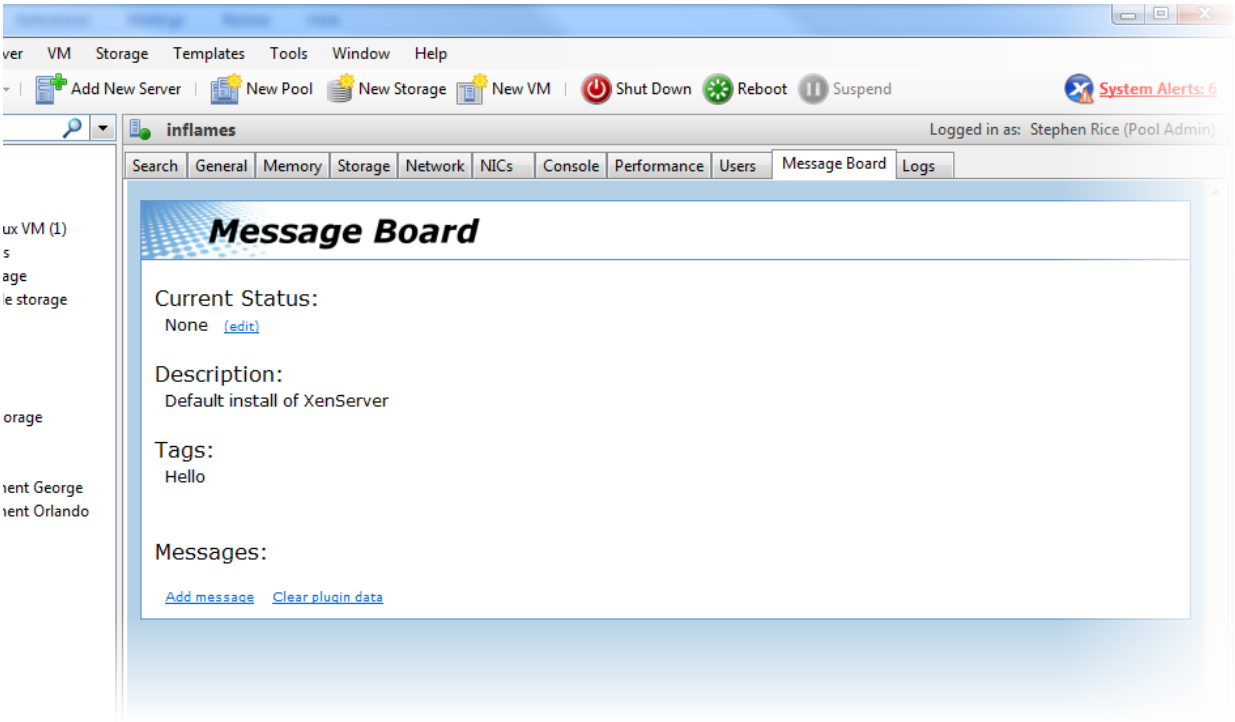
Key	Value	Description	Optional/	Default

			Required	
-	• -	*[All attributes inherited from feature]*	-	-
menu	• file • view • pool • server • vm • storage • templates • tools • help	The main XenCenter menu you would like this to appear under.	Required	-
contextmenu	• none • pool • server • vm • storage • template • folder	An additional context menu you would like this menu item to appear under. Unless you set 'none' the item will already be present on the context menu that relates to the 'menu' attribute (if such a context menu exists).	Optional	[value for menu]

Features: TabPage

Tab page features load a URL to display as an extra tab inside XenCenter. In this way they can be used to allow access to web management consoles or to add additional user interface features into XenCenter.

Note: All local HTML and JavaScript examples in this section are using the modified JQuery libraries for RPC calls through XenCenter as well as the vanilla JQuery base library v1.3.2



Features: TabPage XML Attributes

Important: The inherited feature attribute 'name' is required and your plugin will not load unless it is set

Important: The 'url' attribute is required and your plugin will not load unless it is set

Key	Value	Description	Optional/ Required	Default
-	• -	*[All attributes inherited from feature]*	-	-
url	• [string]	The local or remote URL where the HTML page to load can be found.	Required	-
context-menu	• true • false	Whether you would like the context menu for this html page to be enabled.	Optional	false
xencenter-only	• true • false	If set, this tabpage will appear when the XenCenter node is selected in the resource list and nowhere else.	Optional	false
relative	• true • false	If set, the 'url' attribute will be interpreted as relative to the XenCenter	Optional	false

		install directory.		
help-link	• [string]	The URL to launch in a separate browser when the user requests for help on the tab page.	Optional	-
credentials	• true • false	Indicates that the webpage is using scripting and wishes to use XenCenter's session credentials to interact with the server. This sets window.external.SessionUuid and window.external.SessionUrl for scripting access. WARNING: BY EXPOSING THESE VARIABLES YOU ARE ALLOWING EXTERNAL WEBPAGES ACCESS TO YOUR XENSERVER	Optional	false
console	• true • false	Indicates that this tab page is meant to replace the standard XenCenter console.	Optional	false

Features: TabPage Javascript API – Introduction

Using some modified JQuery libraries to pass XML-RPC calls through XenCenter it is possible for your tab page to communicate with XenServer using JavaScript. XenCenter provides a scripting object which contains the following public variables:

- SessionUuid
- SessionUrl
- SelectedObjectType
- SelectedObjectRef

These can be accessed through the `window.external` object in JavaScript and used to make XenServerAPI calls:

```
//  Retrieves the other config map for the currently selected XenCenter object and passes it on to the
callback function

function GetOtherConfig(Callback)

{

        var tmprpc;
```

```
        function GetCurrentOtherConfig()
        {
                var toExec = "tmprpc." + window.external.SelectedObjectType + ".get_other_config(Callback,
window.external.SessionUuid, window.external.SelectedObjectRef);";
                eval(toExec);
        }
        tmprpc= new $.rpc(
                "xml",
                GetCurrentOtherConfig,
                null,
                [window.external.SelectedObjectType + ".get_other_config"]
        );
}
```

In this example we create an RPC object using 4 parameters:

1. Notifying that the RPC call is carrying XML (as opposed to JSON)
2. The function to execute (`GetCurrentOtherConfig`) which fires off an API call - the modified JQuery library packages up the API call as an XML-RPC request and hands it to XenCenter. Notice that the function name for the callback is passed as an additional first parameter to the API call.
3. We don't specify a version number for the XML (`null`) which is interpreted as 1.0.
4. We pass a description of the API call we want to make inside `GetCurrentOtherConfig` so the appropriate objects are created for the function to access.

Features: TabPage Javascript API – Required Functions

Important: It is required that you define a `RefreshPage` function in your JavaScript.

XenCenter will call this function every time it reloads the HTML page or adjusts the variables on the scripting object. Your code should be structured so that the `RefreshPage` function can easily tear down and rebuild the state of the page:

```
$(document).ready(RefreshPage);


function RefreshPage()
{
        // hide the error div and show the main content div
        $("#content").css({"display" : ""});
        $("#errorContent").css({"display" : "none"});
        $("#errorMessage").html("");
        RefreshMessagesAndStatus();
        RefreshDescription();
        RefreshTags();
}
```

Setting the function to be called at `$(document).ready()` ensures there are no race conditions between XenCenter signaling that the page should refresh and the page itself being ready to receive these requests.

Features: TabPage Javascript API – Receiving XenCenter Callbacks

When you make an RPC object and get XenCenter to pass through an API call to XenServer you specify a callback function. When the XML-RPC request returns from XenServer the callback function is invoked by XenCenter, passing in a JSON object which contains the result as a parameter. Look at `RefreshDescription` below:

```
// The result object of any xmlrpc call to the server contains:
// - a result field which indicates whether it was succesfull or not,
// - a value field containing any returned data in json
// - an error description field containing any error information
// This function checks for success, displays any relevant errors, and returns a json object that
corresponds to the value field
function CheckResult(Result)
{
        var myResult=Result.result;
        if(myResult.Status=="Failure")
        {
                var message=myResult.ErrorDescription[0];
                for(var i=1; i<myResult.ErrorDescription.length; i++)
                {
                        message+=","+myResult.ErrorDescription[i];
                }
                $("#content").css({"display" : "none"});
                $("#errorContent").css({"display" : ""});
                $("#errorMessage").html(message);
                return;
        }
        if (myResult.Value == "")
        {
                return;
        }
        myResult = eval("("+myResult.Value+")");
        return myResult;
}
// DESCRIPTION UPDATE SECTION
// This pair of methods chain to retrieve the description field from the server and display it. There is no
writing to the description field on the server.
function RefreshDescription()
{
        var tmprpc;
        function RetrieveDescription()
        {
```

```
                var toExec = "tmprpc." + window.external.SelectedObjectType +
".get_name_description(ShowDescription, window.external.SessionUuid, window.external.SelectedObjectRef);";
                eval(toExec);
        }
        tmprpc= new $.rpc(
                "xml",
                RetrieveDescription,
                null,
                [window.external.SelectedObjectType + ".get_name_description"])
}


function ShowDescription(DescriptionResult)
{
        var result = CheckResult(DescriptionResult);
        if (result == null)
        {
                $("#descriptionText").html("None");
                return;
        }
        $("#descriptionText").html(result);
}
```

Features: TabPage Replacement Consoles

This feature allows you to specify that your tab page feature should replace the standard console tab page in XenCenter. It is often used when a VM has its own web interface and the standard console tab page does not need to be seen.

To activate this feature, add the attribute `console="True"` to the `TabPageFeature` tag in your configuration file.

If the web page you have specified in your tab page feature cannot be reached by XenCenter, then the standard console tab page will be returned and your tab page feature will be hidden. If the tab page feature can be reached later on, it will be automatically restored, and the standard console tab page hidden.

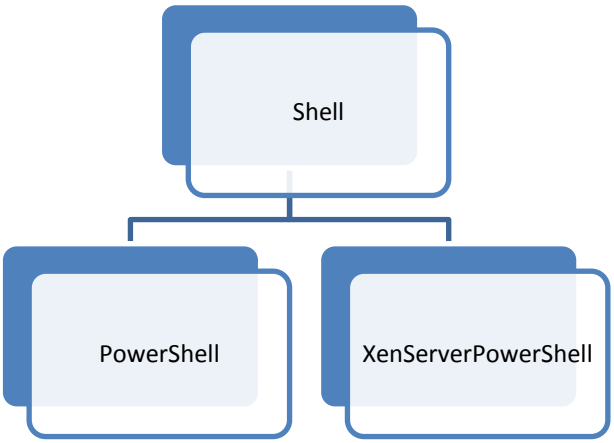
Commands

In your configuration file, each MenuItem feature should have a single command as a child. They detail which executable or script to run when the user clicks the MenuItem.

In this version of the XenCenter plugin specification there are three types of command:

- Shell
- PowerShell
- XenServerPowerShell

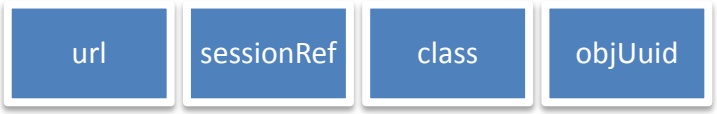
Powershell and XenServerPowerShell are extensions of Shell and inherit all the properties of Shell. However they both have additional features which make it easier to run powershell scripts, for example XenServerPowerShell commands automatically load the XenServer PowerShell Snapin before executing.



Commands: Parameter Sets

A parameter set is a collection of 4 parameters that are used to describe which items are selected in the XenCenter resource list when your command is executed.

While each command has it's own way of receiving parameters from XenCenter, the parameters will always be the same. They are delivered in sets of four which describe the selection in the XenCenter resource list:



Two of the parameters are used to allow communication to the relevant server:

- The `url` parameter indicates the address of the applicable standalone server or pool master
- The `sessionRef` parameter is the session opaque ref that can be used to communicate with this server

Two of the parameters are used to describe which specific object is selected:

- The `class` parameter is used to show the class of the object which is selected in the resource list
- The `objUuid` parameter is the uuid of this selected object

Example: If you were to select both the local SR from a standalone server (Server A) and the pool node from a separate pool (Pool B), you would end up 2 parameter sets being passed into your plugin:

In general you will receive one parameter set per object selected in the tree view, with two exceptions.

- Selecting a folder will give add a parameter set per object in the folder, not the folder itself
- Selected the XenCenter node will add a parameter set for each stand alone server or pool that is connected, with the `class` and `objUuid` parameters marked with the keyword ‘blank’

In this way, if the XenCenter node is selected you will be passed the necessary information to perform actions on any of the connected servers. However, selecting this node provides no contextual information about what the user wishes to target, so the ‘blank’ keyword is used for the parameters which would identify specifically what is selected.

Example: If you were connected to Server A and Pool B from the previous example, and you selected both the XenCenter node and the local storage on Server A you would get the following parameter sets:

https://servera:443/	OpaqueRef:19969d9e-794f-0eb0-2522-950c5ac35d31	blank	blank
https://masterofpoolb:433/	OpaqueRef:c473046d-9836-48ab-85a3-11c5a7c9ca51	blank	blank
https://servera:443/	OpaqueRef:19969d9e-794f-0eb0-2522-950c5ac35d31	SR	1b45aa8f-0c32-4c5d-e7b7-8add8007d04d

Commands: Shell

Shell commands are the most generic command type and launch executables, batch files and other files which have a registered windows extension.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XenCenterPlugin PUBLIC "-//XENCENTERPLUGIN//DTD XENCENTERPLUGIN1//EN" "xencenter-1.dtd">
<XenCenterPlugin
        xmlns="http://www.citrix.com/XenCenter/Plugins/schema"
        version="1"
        plugin_version="1.0.0.0">
        <MenuItem
                name="hello-menu-item"
                menu="file"
                serialized="obj">
                <Shell
                        filename="Plugins\Citrix\HelloWorld\HelloWorld.bat"
                        window="true"
                        log_output="true"
                        dispose_time="0"
                        param="{$type}"
                />
        </MenuItem>
</XenCenterPlugin>
```

Commands: Shell Parameters

For Shell commands the parameter sets are passed through as command line parameters to the batch file or executable. Any additional parameters supplied using the 'param' xml attribute (see below) will be first in the list of parameters, followed by sets of four command line parameters representing each parameter set.

Commands: Shell XML Attributes

Note: The filename attribute is required and your plugin will not load unless it is set

Key	Value	Description	Optional/ Required	Default	Accepts Placeholders
filename	• [string]	The file that you wish to execute, for example an executable or a windows batch file. This is a relative path from your XenCenter install directory.	Required	-	True
window	• True • False	Whether to start the process in a window or run it in the background.	Optional	true	-
log_output	• True • false	Redirects the Standard Output and Standard Error streams of the plugin process to the XenCenter log file.	Optional	false	-
param	• [string]	A comma separated string of extra parameters to pass to the process. You can pass a parameter with spaces by encasing it in XML escaped quotes (")	Optional	-	True
dispose_time	• [float]	The grace period XenCenter should give the plugin after it has requested it to cancel. After this period, XenCenter assumes the plugin has hung and warns the user.	Optional	20.0	-
required_methods	• [string]	A comma separated string of api calls the plugin wishes to run. XenCenter blocks the plugin launch and warns	Optional	-	False

		the user when these requirements are not met due to a role restriction under RBAC.			
required_method_list	• [string]	The name of a list of methods called out in the MethodList node (child of XenCenterPlugin). If required_methods is set, this is ignored. Added in version 5.6 Feature Pack 1.	Optional	-	False

[1] See the section on RBAC protection for more details

Commands: PowerShell

Powershell commands specifically target powershell scripts and have several enhancements over a basic Shell command to help you access the various parameters XenCenter wishes to pass to your plugin.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XenCenterPlugin PUBLIC "-//XENCENTERPLUGIN//DTD XENCENTERPLUGIN1//EN" "xencenter-1.dtd">
<XenCenterPlugin
        xmlns="http://www.citrix.com/XenCenter/Plugins/schema"
        version="1"
        plugin_version="1.0.0.0">
        <MenuItem
                name="hello-menu-item"
                menu="file"
                serialized="obj"
                <PowerShell
                        filename="Plugins\Citrix\HelloWorld\HelloWorld.ps1"
                        debug="true"
                        window="true"
                        log_output="true"
                        dispose_time="0"
                        param="{$type}"
                        function="Write-Output {$type}; read-host '[Press Enter to Exit]'"
                />
        </MenuItem>
</XenCenterPlugin>
```

Commands: PowerShell Object Information Array

Information regarding the target items selected in the XenCenter resource list are stored in a powershell variable for easy access by your script. Inside the `$objInfoArray` variable you will find an array of hashmaps, each representing a parameter set. Use the following keys to access the parameters in each hashmap:

- 'url'
- 'sessionRef'
- 'class'
- 'objUuid'

```
 [reflection.assembly]::loadwithpartialname('system.windows.forms')
foreach ($objInfo in $objInfoArray)
{
        $outputString = "url={0}, sessionRef={1}, objName={2}, objUuid={3}" -f $objInfo["url"],
$objInfo["uuid"], $objInfo["class"], $objInfo["objUuid"]
        [system.Windows.Forms.MessageBox]::show("Hello from {0}!" -f $outputString)
}
```

Commands: PowerShell Extra Parameter Array

Any additional parameters you define using the 'param' XML attribute inherited from Shell are stored in the `$ParamArray` variable as a simple array.

```
[reflection.assembly]::loadwithpartialname('system.windows.forms')
foreach ($param in $ParamArray)
{
        [system.Windows.Forms.MessageBox]::show("Hello from {0}!" -f $param)
}
```

Commands: PowerShell XML Attributes

Note: The filename attribute inherited from Shell is required and your plugin will not load unless it is set to point to a powershell script

Key	Value	Description	Optional/ Required	Default	Accepts Placeholders
-	-	*[All attributes inherited from Shell]*	-	-	-
debug	• true • false	Enables debugging output that traps and details any uncaught exceptions. It is highly recommended you set window=true if debug is	Optional	false	-

		enabled.			
function	• [string]	Executes the provided string as powershell code after the main script has finished executing.	Optional	-	True

Commands: XenServerPowerShell

In addition to the features provided by the PowerShell Command, the XenServerPowerShell Command loads the XenServerPowershellSnapIn before executing your target powershell script.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XenCenterPlugin PUBLIC "-//XENCENTERPLUGIN//DTD XENCENTERPLUGIN1//EN" "xencenter-1.dtd">
<XenCenterPlugin
        xmlns="http://www.citrix.com/XenCenter/Plugins/schema"
        version="1"
        plugin_version="1.0.0.0">
        <MenuItem
                name="hello-menu-item"
                menu="file"
                serialized="obj"
                <XenServerPowerShell
                        filename="Plugins\Citrix\HelloWorld\HelloWorld.ps1"
                        debug="true"
                        window="true"
                        log_output="true"
                        dispose_time="0"
                        param="{$type}"
                        function="Write-Output {$type}; read-host '[Press Enter to Exit]'"
                />
        </MenuItem>
</XenCenterPlugin>
```

Commands: XenServerPowerShell Initialization Details

To see precisely what set up is done to prepare your powershell environment for communicating with XenServer, see the Initialize-Environment script in your XenServerPSSnapin install directory. In brief, when your target script begins executing:

- All the XenServer cmdlet aliases will be initialized
- The global session variable will be initialized to store your session information

Commands: XenServerPowerShell Parameters

The parameter sets and additional parameters can be accessed through the `$objInfoArray` and the `$ParamArray` variables as detailed in the previous PowerShell Command section.

Commands: XenServerPowerShell XML Attributes

Important: The filename attribute inherited from Shell is required and your plugin will not load unless it is set to point to a powershell script

Key	Value	Description	Optional/ Required	Default	Accepts Placeholders
-	• -	*[All attributes inherited from Shell]*	-	-	-
debug	• true • false	Enables debugging output that traps and details any uncaught exceptions. It is highly recommended you set window=true	Optional	false	-
function	• [string]	Executes the provided string as powershell code after the main script has finished executing.	Optional	-	True
version_min	• [string]	Not Implemented (CA-40580)	-	-	-
version_max	• [string]	Not Implemented (CA-40580)	-	-	-

Commands: Preparing for RBAC

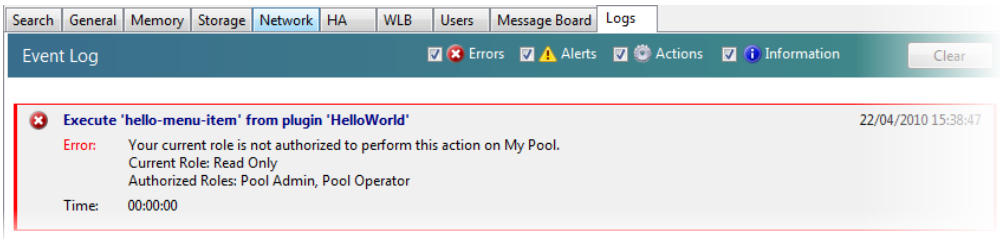
By defining the methods that it will require in your configuration file a command can be prepared for the situation when XenCenter is connected to a server which is using Role Based Access Control.

If the user does not have permission to execute an API call on a server due to RBAC, then the call will fail with an RBAC_PERMISSION_DENIED exception. You can handle these exceptions from within the plugin (examine the ErrorDescription field on the response for details) or you can ask XenCenter to ensure that the user can execute all possible commands you might need before the plugin is run:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XenCenterPlugin PUBLIC "-//XENCENTERPLUGIN//DTD XENCENTERPLUGIN1//EN" "xencenter-1.dtd">
<XenCenterPlugin
        xmlns="http://www.citrix.com/XenCenter/Plugins/schema"
        version="1"
        plugin_version="1.0.0.0">
        <MenuItem
                name="Hello Exe World"
                menu="file"
                serialized="obj"
                description="The worlds most friendly plugin, it loves to say hello">
                <Shell
                        filename="Plugins\Citrix\HelloWorld\HelloWorld.exe"
                        required_methods="host.reboot, vm.start"
                />
        </MenuItem>
</XenCenterPlugin>
```

The `required_methods` attribute accepts a comma separated list of API calls in the format: `object.method`.

If the user is operating on an RBAC enabled server, XenCenter will check that they can execute all of these API calls on their current role. If they can't, the plugin is not launched and an error displayed:



Commands: Preparing for RBAC - Key White Lists

Important: Key white lists were first exposed for plugin use in version 5.6 Feature Pack 1

You will be unable to use the white list for modifying map keys in version 5.6

Important: These white lists apply to advanced XenCenter keys. You should only modify them if you know what you are doing.

In general, when operating under RBAC your role restricts you to modifying the `other-config` map on objects which are directly relevant to your role. For example a VM Admin can modify the `other-config` map on a VM, but it cannot modify the `other-config` map on a server.

Some specific keys have been white listed to all roles above read-only to allow XenCenter to set some advanced keys on `other-config` maps that would otherwise be inaccessible to the user's role:

Target object	Key
VDI SR network host VM pool	XenCenter.CustomFields.*
VDI SR Network host VM pool	folder
pool	EMPTY_FOLDERS
task	XenCenterUUID
task	applies_to
network	XenCenterCreateInProgress

You can enter these checks into your method list with the following syntax:

```
<MethodList name="methodList1">
pool.set_other_config/key:folder,
pool.set_other_config/key:XenCenter.CustomFields.*
</MethodList>
```

Commands: Placeholders

By leaving placeholders in your strings, XenCenter can call different functions, URLs or provide different parameters based on what object is selected in the resource list.

When an XML attribute is marked as being able to accept placeholders you can leave wildcards for XenCenter to fill in based on the properties of the object that is selected in the resource list.

Placeholder	Description
{$type}	The type of the selected object, e.g., VM, Network
{$label}	The label of the selected object
{$uuid}	The UUID of the selected object, or the full pathname of a folder
{$description}	The description of the selected object
{$tags}	Comma-separated list of the tags of the selected object
{$host}	The host name
{$pool}	The pool name
{$networks}	Comma-separated list of the names of the networks attached to the object
{$storage}	Comma-separated list of the names of the storage attached to the object
{$disks}	Comma-separated list of the types of the storage attached to the object
{$memory}	The host memory, in bytes
{$os_name}	The name of the operating system that a VM is running
{$power_state}	The VM power state, e.g. Halted, Running
{$virtualisation_status}	The state of the pure virtualization drivers installed on a VM
{$start_time}	Date and time that the VM was started

{$ha_restart_priority}	The HA restart priority of the VM
{$size}	The size in bytes of the attached disks
{$ip_address}	Comma-separated list of IP addresses associated with the selected object
{$uptime}	Uptime of the object, in a form such as '2 days, 1 hour, 26 minutes'
{$ha_enabled}	true if HA is enabled, false otherwise
{$shared}	Applicable to storage, true if storage is shared, false otherwise
{$vm}	Comma-separated list of VM names
{$folder}	The immediate parent folder of the selected object
{$folders}	Comma-separated list of all the ancestor folders of the selected object

If the user has selected more than one target for the plugin (by multiselect or by selecting a folder) then it is not possible for XenCenter to know which object to use for the placeholder context.

It is possible that in the future the placeholder logic will be extended to allow the plugin author to indicate which placeholders should be filled with which objects in a multi target scenario. Currently in a multi target scenario all placeholders will be substituted with the keyword 'multi_target', so it will at least be possible to detect this situation.

Additionally, if there has been an error filling in a particular placeholder then the keywork 'null' will be substituted in to indicate this. One example of where you would see this is if the XenCenter node was selected, which has no object properties to fill in.

Example: A community group adds their HTML help guides into a XenCenter tab based on which object is selected

```
<XenCenterPlugin xmlns="http://www.citrix.com/XenCenter/Plugins/schema" version="1"
plugin_version="1.0.0.0">
        <TabPage
                name="Extra Support"
                url="http://www.extra-help-for-xencenter.com/loadhelp.php?type={$type}"
        />
</XenCenterPlugin>
```

Resources and Internationalization

In the text below, *<plugin_name>* is the value of the Plugin name attributes and *<entry_name>* is the value of the MenuItem/TabPage name attributes.

The following resources will be read from *<plugin.resources.dll>*:

- *<plugin_name>*.description – Shown in the plugins dialog.
- *<plugin_name>*.copyright – Vendor copyright statement. Shown in the plugins dialog.
- *<plugin_name>*.link – Link to vendor's webpage. Shown in the plugins dialog.
- *<entry_name>*.label – The menu entry label.
- *<entry_name>*.description – Shown in the plugins dialog.
- *<entry_name>*.icon – The icon to use in the menu entry. Should be a 16x16 PNG. May be omitted.
- *<entry_name>*.tooltip – The tooltip to use when the menu entry is disabled. May be omitted.

To create the resources file:

- Create a RESX file containing the appropriate strings (this can be done in any project in Visual Studio: e.g., console project).
- Open up Visual Studio cmd prompt and navigate to the resx's directory
- Run `ResGen.exe <plugin_name>.resx`. (ResGen.exe is found in C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin).
- Run `al.exe /t:lib /embed:<plugin_name>.resources /out:<plugin_name>.resources.dll`. (al.exe is also found in C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin).
- If you wish to compile additional resource DLLs for any specific cultures edit the resx file appropriately and run these commands again with an additional /culture argument in the al.exe command specifying the two letter culture string: e.g., for Japanese, run `al.exe /t:lib /embed:<plugin_name>.resources /culture:ja /out:<plugin_name>.resources.dll`
- Place the invariant culture `<plugin_name>.resources.dll` into the *<plugin_dir>* folder (with `<plugin_name>.xcplugin.xml` etc)
- Other cultures should be as follows: `<plugin_dir>\<culture>\<plugin_name>.resources.dll` where *<culture>* is the two-letter ISO culture name

Unless stated otherwise, all of these entries are mandatory. If any are missing, then the problem will be logged, and the menu option will be disabled.

Deploying

Each plugin is installed into:
<XenCenter_install_dir>\Plugins*<organization_name>**<plugin_name>*.

Notes:

- Plugins should expect *<org_root>* to be read-only at runtime.
- By default *<XenCenter_install_dir>* is `C:\Program Files\Citrix\XenCenter`.
- *<organization_name>* is the name of the organization or individual authoring the plugin.
- *<plugin_name>* is the name of the plugin.
- The author's installation directory (*<XenCenter_install_dir>*\Plugins*<organization_name>*) is known in this specification as *<org_root>*.
- *<XenCenter_install_dir>* can be looked up in the registry. If XenCenter was installed as "just me" the location of the XenCenter install directory can be found at `HKEY_CURRENT_USER\SOFTWARE\Citrix\XenCenter\InstallDir`. If XenCenter was installed for all users this key is under `HKEY_LOCAL_MACHINE`.

In *<org_root>**<plugin_name>* there should be:

- A plugin declaration file, named *<plugin_name>*`.xcplugin.xml`.
- A satellite assembly for resources, named *<plugin_name>*`.resources.dll`.
- Anything else that the plugin needs for its proper functions.

Other than as specified in this document, XenCenter ignores the contents of *<org_root>*. Plugin authors are free to install whatever content they need within their own *<org_root>*. You may, for example, wish to have libraries or graphics that are shared between all their plugins, in which case *<org_root>* (or a shared subdirectory of it) would be a good place for these.

The same freedom applies to *<org_root>**<plugin_name>*; material in there that isn't recognized by XenCenter will be ignored.

The *<XenCenter_install_dir>*`\Plugins` directory is scanned when XenCenter starts and plugins that are found will be loaded. The directory can be re-scanned by restarting XenCenter or by pressing the `Re-Scan Plugin Directory` button on the XenCenter plugins dialog. Any subdirectory that does not contain *<plugin_name>*`.xcplugin.xml` will be silently ignored.