

# Getting Started with VC Plug-ins

## Extending the VI Client through VirtualCenter 2.5

---

With the release of VirtualCenter 2.5, VMware offers third-party developers and partners the ability to extend the VMware Infrastructure Client (VI Client) with their own product-specific menu selections, views, tabs, and toolbar icons, to provide access to external, Web-based functionality.

These VirtualCenter Server extensions, or *VC Plug-ins*, comprise the set of configuration files, URLs, icons, and Web-server-hosted resources that work together to display extended menu items, icons, and other user interface (UI) items in the VI Client and provide access to the external functionality.

Adding a plug-in to the VI Client is relatively simple—the only requirement for the extended functionality is that it be available from a Web server, using a standard URL that can be accessed directly from your end-users' desktop machines.

This technical note provides an overview of the VC Plug-in architecture and helps you get started adding your own extensions to the VI Client. It includes these topics:

- [Introducing VC Plug-ins](#)
- [Developing a VC Plug-in](#)
  - [Step 1. Develop and Deploy the Application](#)
  - [Step 2. Identify Areas of the VI Client to Extend](#)
  - [Step 3. Create the Configuration File](#)
  - [Step 4. Register the Extension with VirtualCenter Server](#)
- [Managing VC Plug-ins](#)
- [Best Practices](#)

The steps described in this technical note comprise a one-time configuration and setup process. Once all the steps are complete, the extension will be available in the VI Client.

---

**NOTE** The technique for extending the VI Client discussed in this technical note is experimental, and may change in future releases.

---

## Introducing VC Plug-ins

VC Plug-ins enable a wide range of integration and extension scenarios. These are a few examples of what you can do using VC Plug-ins:

- Display a static Web page. For example, you can have the VI Client navigate to your company Web site, by means of a custom button added to the VI Client toolbar.
- Display a dynamic Web page that refreshes data with each access. See [Example 1, "Configuration File \(Minimal\),"](#) on page 7, for an example configuration file that displays the Google maps page.
- Obtain listings of managed ESX Server host systems.

- Obtain information about the inventory—host systems, virtual machines, compute resources, and so on.
- Retrieve performance statistics and information about events.
- Manage the entire inventory, for example, add a new host to the inventory, power-on virtual machines, power-off virtual machines, and so on.
- Act on information retrieved from VirtualCenter Server. For example, obtain the DNS name of a selected host and formulate a URL pointing to your own management system on that host.

Developing and deploying VC Plug-ins requires Web-server based components. The application must run on a Web server and must be implemented using CGI scripting, Java Servlets, ASP.NET, or other standard Web-development approaches. The URL to the application is used at runtime to populate the VI Client with menus, toolbar icons, and the like, which ultimately provide access to your extended functionality.

This Web-based approach to VC Plug-ins (extensions) offers several advantages, including:

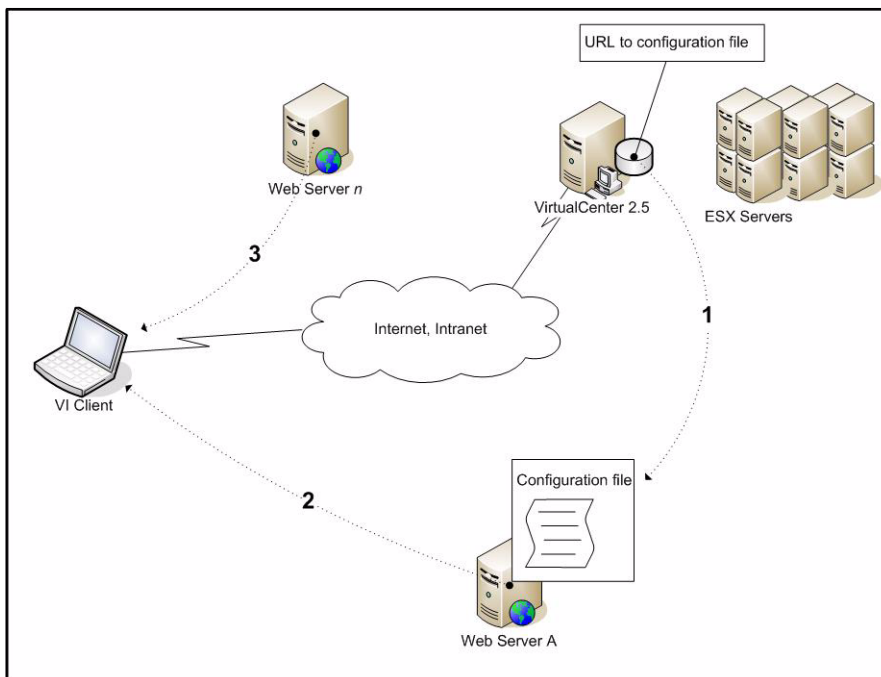
- Existing Web-based applications or any Web site can be integrated with VI Client, quickly and easily.
- The integration (with VI Client) is loosely coupled: the application can be invoked from within VI Client, but can also run as a standard Web application, inside any Web browser.

## System Architecture

Extending the VI Client is accomplished by using the ExtensionManager (supported on VirtualCenter Server 2.5) in conjunction with a configuration file and your Web-server based management application.

Figure 1 provides an overview of the various components. (Assuming that you are using the Windows platform for your Web servers, the Web-server-based components can all run on the same hardware as the VirtualCenter Server system.)

**Figure 1.** How It Works



Here's a brief overview of how the components shown in Figure 1 populate the VI Client and provide access to external (non-VirtualCenter) functionality.

- 1 The user connects to the VirtualCenter through the VI Client.

The VirtualCenter Server's ExtensionManager sends to the VI Client the URL pointing to the location of the configuration file.

- 2 The VI Client obtains the configuration file from the Web server.
- 3 The VI Client is populated with the UI items—menus, buttons, and so on—from the location (or locations) specified in the configuration file.

When the user subsequently selects an extended menu or button, the VI Client connects to the Web server specified in the configuration file, passing current context information in the URL string:

- **sessionId**—String value obtained by VI Client after logging in to VirtualCenter Server. The sessionId associates access to the VC Plug-in application to the same user (and his or her privileges) as the VI Client's current login user.
- **moref**—Managed object reference to a managed object that has been selected from the inventory. The moref type and value are separated by a colon (:). For example, moref=VirtualMachine:16.
- **serviceUrl**—Server name and path to the Web services API hosted on VirtualCenter Server, as a complete URL. For example, serviceUrl=https://localhost/sdk or serviceUrl=https://serverFQDN/sdk
- **locale**—The name of the locale from the VI Client, to be used for localizing content from the Web server, if necessary. For example, locale=en, locale=ja, or locale=de sets the server's locale to English, Japanese, and German, respectively.

The VI Client passes its serviceUrl, sessionId, and locale to the Web application, so the Web application doesn't require an additional authentication process. In addition, a moref argument (the managed object reference) is passed when the VC Plug-in is associated with a specific inventory item—a managed entity, such as a VirtualMachine.

For example, assume a VC Plug-in extends the functionality available to virtual machine operations with a script that automates power-down. The URL specified in the configuration file might be as follows:

```
http://dev:8000/vmAction.cgi?cmd=powerOn
```

At runtime, when the VI Client user activates the associated menu item or icon, the URL sent to the VirtualCenter Server from the VI Client might look like this:

```
http://dev:8000/vmAction.cgi?cmd=powerOn&moref=VirtualMachine:16&sessionId=9241E7B8-A37B-4264-A8D1-945628F9E0D6&locale=en&serviceUrl=https://localhost/sdk
```

Central to VC Plug-ins is the ExtensionManager managed object, supported on VirtualCenter Server 2.5. The next section discusses this managed object in a bit more detail.

## VirtualCenter Server ExtensionManager

The ExtensionManager is a managed object type, introduced in VirtualCenter Server 2.5. ExtensionManager has a single property, extensionList, which holds an array of extensions (the Extension data object type) that have been registered with it.

Figure 2 shows a partial class hierarchy diagram of the ExtensionManager managed object.

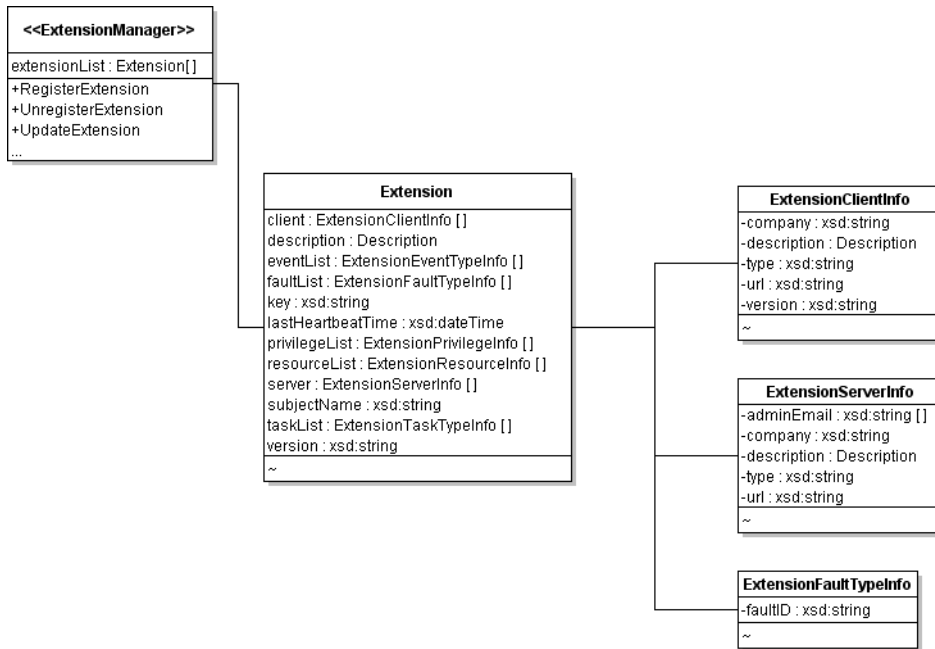
**Figure 2.** Overview of ExtensionManager Managed Object and Associated Data Objects

Figure 2 shows the three operations primarily used by VC Plug-in developers, specifically:

- RegisterExtension
- UnregisterExtension
- UpdateExtension

Complete information about these operations, the privileges required to invoke them, and other details are in the VI API Reference 2.5 at:

<http://pubs.vmware.com/vi-sdk/visdk250/ReferenceGuide/vim.ExtensionManager.html>

**Figure 3.** VI API Reference for ExtensionManager Managed Object Type

The screenshot shows the VI API Reference web page for the ExtensionManager Managed Object Type. The page includes a navigation menu on the left, a main content area with tabs for Managed Object Types, Data Object Types, Local Properties, and Local Methods, and a detailed description of the managed object type.

**Managed Object - ExtensionManager**

*Property of* ServiceContent  
*See also* Extension  
*Since* VI API 2.5

**Managed Object Description**

This managed object type provides directory and basic management services for all registered extensions. Clients use the ExtensionManager, available in ServiceInstance, to access extension objects.

**Properties**

NAME	TYPE	DESCRIPTION
extensionList	Extension[]	The list of currently registered extensions.

May not be present. Required privilege: System.View

**Methods**

**METHODS DEFINED IN THIS MANAGED OBJECT**

FindExtension, GetPublicKey, RegisterExtension, SetPublicKey, UnregisterExtension, UpdateExtension
--

**FindExtension**

Returns extension with the given key, if any.

**Required Privileges**

System.View

To get started, you will need to use only the RegisterExtension operation. (You'll see how to do this in "Step 4. Register the Extension with VirtualCenter Server" on page 10.)

## Developing a VC Plug-in

Developing a VC Plug-in is a four-step process:

- 1 Develop and deploy your Web-based application (if it does not exist). “[Step 1. Develop and Deploy the Application](#),” discusses some of the requirements about how to host your application and describes some of the possibilities for integration.
- 2 Identify the areas of the UI to which to add menu selections, icons, and the like. See “[Step 2. Identify Areas of the VI Client to Extend](#)” on page 5 for details.
- 3 Create the XML-based extension configuration file. The extension file must conform to a specific XML schema defined by VMware. See “[Step 3. Create the Configuration File](#)” on page 7 for details.
- 4 Register the extension with VirtualCenter Server using the VI API, specifically, the RegisterExtension operation available through the ExtensionManager managed object. (“Registering the extension” involves passing a URL for the management application to VirtualCenter Server.) See “[Step 4. Register the Extension with VirtualCenter Server](#)” on page 10 for details.

### Step 1. Develop and Deploy the Application

The only requirement for extending the VI Client is that the functionality be available on a Web server. The “functionality” can be as sophisticated as a comprehensive administration application, or as simple as static delivery of the home page to your company’s Web site.

You can implement the Web server functionality using the programming language of your choice. For example:

- Java Servlets or Java Server Pages (JSPs)
- Traditional CGI Scripting
- Microsoft Active Server Pages (ASP.NET)
- Static or dynamic HTML pages

---

**NOTE** Java applets may or may not work, depending on the specifics of the deployment.

---

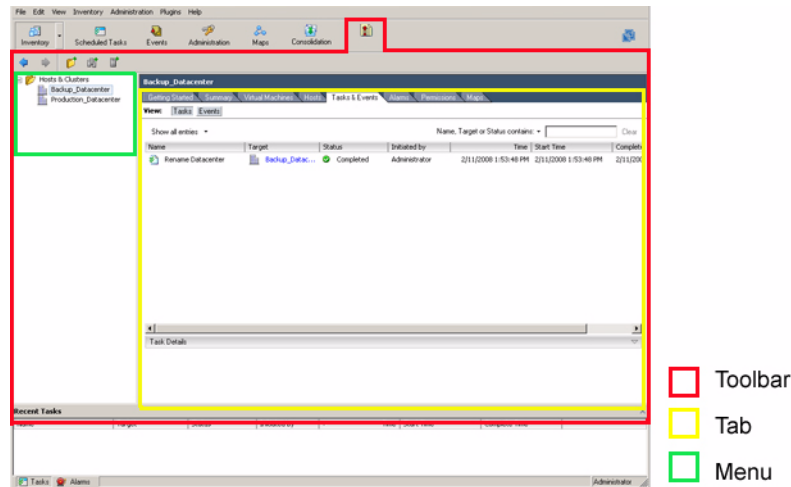
The remaining steps of this technical note assume that you have a Web server or other Web-based functionality available to use.

### Step 2. Identify Areas of the VI Client to Extend

Assuming you have the Web-server-based application or Web page ready to integrate, the next step is to identify where and how to extend the VI Client.

For example, should access be provided by means of a drop-down menu, from a particular inventory item type, or via an icon in the Toolbar? The VI Client can be extended in several distinct ways, specifically:

- Icon buttons can be added to the toolbar (for example, the Hyperlink icon shown in [Figure 4](#)).
- Context-dependent menu items can be added to the VI Client under the inventory tree (the area shown in green, in [Figure 4](#)).
- Tabs can be associated with specific inventory objects (managed entities).

**Figure 4.** VI Client User Interface Extension Areas

VMware plans to publish user-interface guidelines for extending the VI Client with VC Plug-ins. [Table 1](#) lists the areas of the VI Client that can be extended and the tag name that accomplishes the extension.

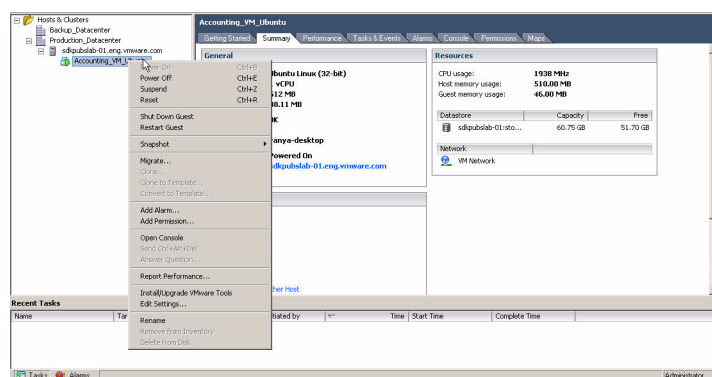
**Table 1.** VI Client Extension Areas Summary

Extension area	Tag	Usage
Menu	command	Set parent attribute to appropriate context-menu or inventory item menu.
Tab	view	Use the view tag to add a new tab to the right-hand pane. Associate the Tab with the appropriate inventory item by setting its parent tag to the managed entity.
Toolbar	view	To add an icon to the toolbar, use the view tag and set its parent attribute to Inventory.Global. Place the icon's image file (for example, .jpg or .gif) on the Web server specified by the URL.

The configuration file, described in detail in [“Step 3. Create the Configuration File,”](#) is the means by which you will specify the particulars.

When creating the GUI components—menu, tab, and other icons, for example—be sure to take into account the look and feel of the VI Client, in general. Your choices regarding fonts, font sizes, color schemes and the like are best considered in the context of the VI Client.

[Figure 5](#) shows the default context menu for the virtual machine managed entity, as an example.

**Figure 5.** Context-menu for Inventory Item

One basic guideline is to keep it simple. Don't overextend the VI Client, particularly on top-level interface plug-ins. The space available for these GUI components is limited, and quickly becomes crowded.

After you've mapped out how you will extend the VI Client, you can create the configuration that will actually do the work of populating the VI Client with UI items.

### Step 3. Create the Configuration File

Having planned out where and how to extend the UI, you can create the configuration file for your plug-in. You can use any text editor to create the configuration file. However, VMware recommends using appropriate tools that validate the XML, such as Altova XML Spy.

The configuration file is an XML file that complies with specific schema defined by VMware. The XML Schema definition (XSD) file for the plug-in configuration file is located on the VMdev.net Web site. However, you don't really need to examine the XSD. Follow the steps in this section to create a configuration file that conforms to the requirements of the XSD:

- 1 Start the configuration file with the `scriptConfiguration` opening tag. Use the `version` attribute of the `scriptConfiguration` tag to specify the plug-in version number. The `version` attribute is used during registration of the plug-in URL. For example:
 

```
<scriptConfiguration version="1.0.0">
```
- 2 Create a `key` tag that will be used by `ExtensionManager` to uniquely identify your extension among possibly dozens of other VC Plug-ins offered by both VMware and third-party developers. The `key` tag accepts a string. VMware recommends following the Java package-naming conventions for `key` tag definitions. For example:
 

```
<key>com.yourcompanyname.your-plugin-name</key>
```
- 3 Add a `description` tag, if you like—it's optional, but VMware suggests that you add a description: the text will display in the Plugin Manager, so users can add it to the VI Client UI.
- 4 Add `view` tags, `menu` tags, or both to define how users will access application features from within the VI Client.
- 5 End the configuration file with the `scriptConfiguration` closing tag (`</scriptConfiguration>`).

**Example 1** shows a minimal configuration file. The `scriptConfiguration` opening and closing tags encompass all the other elements. Of the other element tags shown in **Example 1**, only the `<key>` and `<title>` tags are required.

#### Example 1. Configuration File (Minimal)

---

```
<scriptConfiguration version="1.0.0">
  <key>com.vmware.cde.sdk.demo</key>
  <description>VI Extensibility Demo</description>
  <view parent="Inventory.Datacenter">
    <title locale="en">MapIt</title>
    <url>http://maps.google.com</url>
  </view>
</scriptConfiguration>
```

---

**Example 2** provides a complete example configuration file that defines several different extensions—icons, menus, and so on—without regard to best practices.

**Example 2. Configuration File (Comprehensive)**


---

```

<scriptConfiguration version="1.0" processingMode="0">
  <key>com.vmware.codev.vixserver</key>
  <description>VI Perl Toolkit Web Application</description>

  <!-- Executes the script at the given url -->
  <command parent="Inventory.HostSystem">
    <url display="none">http://10.20.52.226:888/cgi-bin/YourPerlScript.pl?Action=Reconnect+Hosts
    </url>
    <title locale="en">Reconnect Hosts</title>
  </command>

  <!-- Launches the URL in a Web browser -->
  <command parent="Inventory.HostSystem">
    <url display="browser">http://10.20.52.226:888/cgi-bin/HostPerformance.pl?</url>
    <title locale="en">Performance Tool</title>
  </command>

  <!-- Nested Menu plug-ins -->
  <command parent="Inventory.VirtualMachine">
    <title locale="en">VM</title>
    <command>
      <title locale="en">PowerOn All VMs</title>
      <url>http://10.17.156.202:9000/cgi-bin/VmAction.cgi?cmd=powerOn&vmname=All</url>
    </command>
  </command>

  <view parent="Inventory.HostSystem">
    <url>http://10.20.52.226:888/cgi-bin/KarlsToolbox.pl</url>
    <title locale="en">Karl's Toolbox</title>
    <title locale="fr">Karl's Toolbox in French</title>
  </view>

  <!-- Adds a new view to a virtual machine, but only in VM & Templates view -->
  <view parent="Inventory.VirtualMachine" treestyle="Virtual">
    <title locale="en"> Resource Allocation</title>
    <url>http://10.20.52.226:888/cgi-bin/ResourceAllocation.pl</url>
  </view>

  <!-- Adds a new view at the global level using the specified icon -->
  <view parent="Inventory.Global">
    <url>http://10.20.52.226:8080/portal/portal/default</url>
    <title locale="en">Dashboard</title>
    <icon>http://10.20.52.226:8080/portal/dashboard.png</icon>
  </view>

  <!-- Adds a new view to a virtual machine that has a custom attribute defined
  that matches the name-value pair -->
  <view parent="Inventory.VirtualMachine" treestyle="Physical">
    <customAttribute name="Attr1">Liquid VM</customAttribute>
    <title locale="en"> Resource Allocation</title>
    <url>http://10.20.52.226:888/cgi-bin/ResourceAllocation.pl</url>
  </view>
</scriptConfiguration>

```

---

After creating the configuration file, place it on the Web server and location of your choosing. Make a note of this information (you will need it to register the URL to the configuration file location in a client-side script or application (see [“Step 4. Register the Extension with VirtualCenter Server”](#) on page 10.)

**Table 2** summarizes the available element tags with their attributes, if any, and provides usage notes. The tilde (~) in the Attributes column denotes that the element has no attributes.

**Table 2.** Configuration File XML Element Definitions

Element	Description and usage note	Attributes	Description and usage note
scriptConfiguration	Required. Top-level (containing) element. Requires a single key element. Specify description, command, and view elements to add the GUI components of your plug-in to the VI Client.	version	Required. The VI Client accepts or rejects the plug-in components based on version.
key	Required. Unique string that identifies the set of components comprising a specific VC Plug-in. VMware recommends using Java package-naming conventions for keys. For example, <code>&lt;key&gt;com.yourcompany.yourapp&lt;/key&gt;</code>	~	~
description	Optional. Text description of the collection of script configuration elements.	~	~
view	Optional. Use in conjunction with parent attribute to specify: <ul style="list-style-type: none"> <li>■ Addition of a tab to the VI Client main pane</li> <li>■ Placement of icon on the VI Client toolbar</li> </ul> If specified, requires: <ul style="list-style-type: none"> <li>■ Exactly one url element.</li> <li>■ One or more title elements</li> </ul> For sub-menus, nest multiple view elements.	parent	Required. Specifies the parent view for the extension. <ul style="list-style-type: none"> <li>■ Set parent to Inventory.Global to consume the entire VI Client UI when the extension is activated.</li> <li>■ Set parent to Inventory.Global and add an <code>&lt;icon&gt;</code> element that specifies the location of an image file for the button or other UI element to display.</li> </ul> Other available options for the parent attribute include: <ul style="list-style-type: none"> <li>■ Inventory.Admin</li> <li>■ Inventory.Cluster</li> <li>■ Inventory.ComputeResourceFolder</li> <li>■ Inventory.Datacenter</li> <li>■ Inventory.DatacenterFolder</li> <li>■ Inventory.Datastore</li> <li>■ Inventory.Global</li> <li>■ Inventory.HostSystem</li> <li>■ Inventory.Network</li> <li>■ Inventory.ResourcePool</li> <li>■ Inventory.Template</li> <li>■ Inventory.VirtualMachine</li> <li>■ Inventory.VirtualMachineFolder</li> </ul>
		treestyle	Optional. By default, view displays in all appropriate positions on the UI. However, you can use the treestyle attribute to specify treestyle display in the left-hand pane only, under the appropriate inventory item. Options for the treestyle attribute include: <ul style="list-style-type: none"> <li>■ Physical</li> <li>■ Virtual</li> <li>■ Datastores</li> <li>■ Network</li> </ul>

**Table 2.** Configuration File XML Element Definitions

Element	Description and usage note	Attributes	Description and usage note
command	Optional. Defines custom menu plug-in or context menu. If specified, requires: <ul style="list-style-type: none"> <li>■ Exactly one url element.</li> <li>■ One or more title elements</li> </ul> For sub-menus, nest multiple command elements.	parent	Required for command element tag. Menu extension points include: <ul style="list-style-type: none"> <li>■ Inventory.Global</li> <li>■ Inventory.Datacenter</li> <li>■ Inventory.DatacenterFolder</li> <li>■ Inventory.HostSystem</li> <li>■ Inventory.VirtualMachine</li> <li>■ Inventory.Admin</li> <li>■ Inventory.Cluster</li> <li>■ Inventory.Datastore</li> <li>■ Inventory.Network</li> <li>■ Inventory.ResourcePool</li> <li>■ Inventory.VirtualMachineFolder</li> <li>■ Inventory.ComputeResourceFolder</li> <li>■ Inventory.Template</li> <li>■ ContextMenus.Alarm</li> <li>■ ContextMenus.Role</li> <li>■ ContextMenus.Permission</li> <li>■ ContextMenus.ScheduledTasks</li> <li>■ ContextMenus.Tasks</li> </ul>
url	Required. Url comprising a script or location of the application or location of an icon, button, or other UI element.	display	Optional. Fine-tune the action taken on the specified plug-in using one of these settings (default is browser): <ul style="list-style-type: none"> <li>■ browser—Open the URL in a separate Web browser window (instead of inside the VI Client).</li> <li>■ window—Open the URL in a new window inside the VI Client.</li> <li>■ none—Execute the script located at the URL (open no windows). Use in conjunction with command=script.</li> </ul>
title	Required. Text that displays for command and view plug-ins.	locale	Required. Associates the title with a specific locale. Any given locale can have only one title.
icon	Optional. A URL containing the icon (as a .gif file) that will display for a view plug-in.	~	~
customAttribute	Optional. Use to conditionally add a menu or view to the specified managed object.	name value	Name and value attribute must map to a custom attribute defined on the managed object. Currently, the VI Client supports adding customAttribute to virtual machine (vm) and hosts (host) only.

## Step 4. Register the Extension with VirtualCenter Server

After the configuration file has been created and placed at the appropriate Web server location, and all other components, including icons, Web server scripts, and other components are in place, you can register the configuration file's URL with the VirtualCenter Server's ExtensionManager. To do so, you must create a simple client application or script that follows the basic client application pattern required for most any VMware Infrastructure client application, specifically:

- 1 Connect to the VirtualCenter Server. Assuming your server is using the default HTTPS, your code must pass the user name and password that has privileges on the "sdk" path of the server port. Typically, the

Administrator (Windows local account) is the user name, but if the VirtualCenter Server has been installed using a different Windows user account, use that account instead.

- 2 Obtain a ManagedObjectReference to the ServiceContent managed object on the VirtualCenter Server system:

```
ManagedObjectReference serviceRef = new ManagedObjectReference();
serviceRef.setType("ServiceInstance");
serviceRef.setValue("ServiceInstance");
VimPortType service = null;
VimServiceLocator serviceLocator = new VimServiceLocator();
serviceLocator.setMaintainSession(true);
```

- 3 Build up the various data objects—Description, ExtensionServerInfo, and ExtensionClientInfo—that will be used to define the properties of the Extension data object:

```
Description description = new Description();
description.setLabel("Optional Label for Your Plug-in Here");
description.setSummary(descStr);
```

```
ExtensionServerInfo esi = new ExtensionServerInfo();
esi.setUrl(ext_url);
esi.setDescription(description);
esi.setCompany(companyStr);
esi.setType("com.vmware.vim.viClientScripts");
esi.setAdminEmail(new String[] {adminEmail});
```

```
ExtensionClientInfo eci = new ExtensionClientInfo();
eci.setVersion(versionStr);
eci.setDescription(description);
eci.setCompany(companyStr);
eci.setType("com.vmware.vim.viClientScripts");
eci.setUrl(ext_url);
```

---

**NOTE** The ExtensionServerInfo and ExtensionClientInfo must both have “type” set specifically to com.vmware.vim.viClientScripts, as shown above. Other variables will be specific to your extension.

---

- 4 Create the Extension data object, passing to it the Description, ExtensionServerInfo, and ExtensionClientInfo data objects, and defining other required properties for the Extension data object:

```
Extension ext= new Extension();
ext.setDescription(description);
ext.setKey(keyStr);
ext.setVersion(versionStr);
ext.setSubjectName("VC Extensibility");
ext.setServer(new ExtensionServerInfo[] {esi});
ext.setClient(new ExtensionClientInfo[] {eci});
ext.setLastHeartbeatTime(Calendar.getInstance());
```

The Extension data object is complete and ready to be passed to the ExtensionManager. As with other managed objects, you must invoke the operation using a managed object reference.

- 5 Obtain a managed object reference for the ExtensionManager:

```
ManagedObjectReference extMgrMof = serviceContent.getExtensionManager();
```

- 6 Invoke the service interface to register the URL associated with the extension:

```
service.registerExtension(extMgrMof, ext);
```

Note that many details have been deliberately left out of the steps listed.

A Java sample that follows this basic client pattern is listed in [Example 3](#). The sample shown in [Example 3](#) uses placeholders for many string values that are instance specific, so the code cannot run as is. Also, [Example 3](#) does not use proper error handling or other coding best practices. Nonetheless, the code can be used as a basic template for using the ExtensionManager to register a VC Plug-in.

**Example 3.** Using the VI API to Register a VC Plug-in with the ExtensionManager

---

```

import com.vmware.vim25.*;
import java.net.URL;
import java.util.*;
public class RegMyPlugin {
    private ManagedObjectReference _svcRef;
    private VimServiceLocator _locator;
    private VimPortType _service;
    private ServiceContent _sic;
    private String companyStr = "Your Company Name Here";
    private String descStr = "Description of your VC Plug-in Here";
    private String keyStr = "com.FQDN.your-vc-plugin-name-here";
    private String ext_url = "http://path-to-your-plugin-xml-file/here";
    private String adminEmail = "your-admin-email-address-here";
    private String versionStr = "1.0.0";
    private void createServiceRef() throws Exception {
        _svcRef = new ManagedObjectReference();
        _svcRef.setType("ServiceInstance");
        _svcRef.set_value("ServiceInstance");
    }
    private void connLoginReg(String hostName, String userName, String password) throws Exception {
        System.setProperty("axis.socketSecureFactory",
            "org.apache.axis.components.net.SunFakeTrustSocketFactory");
        String url = "https://" + hostName + "/sdk/vim";
        Description description = new Description();
        description.setLabel("Optional Label for Your Plug-in Here");
        description.setSummary(descStr);
        ExtensionServerInfo esi = new ExtensionServerInfo();
        esi.setUrl(ext_url);
        esi.setDescription(description);
        esi.setCompany(companyStr);
        esi.setType("com.vmware.vim.viClientScripts");
        esi.setAdminEmail(new String[] {adminEmail});
        ExtensionClientInfo eci = new ExtensionClientInfo();
        eci.setVersion(versionStr);
        eci.setDescription(description);
        eci.setCompany(companyStr);
        eci.setType("com.vmware.vim.viClientScripts");
        eci.setUrl(ext_url);
        Extension ext = new Extension();
        ext.setDescription(description);
        ext.setKey(keyStr);
        ext.setVersion(versionStr);
        ext.setSubjectName("VC Extensibility");
        ext.setServer(new ExtensionServerInfo[] {esi});
        ext.setClient(new ExtensionClientInfo[] {eci});
        ext.setLastHeartbeatTime(Calendar.getInstance());
        createServiceRef();
        _locator = new VimServiceLocator();
        _locator.setMaintainSession(true);
        _service = _locator.getVimPort(new URL(url));
        _sic = _service.retrieveServiceContent(_svcRef);
        if (_sic.getSessionManager() != null) {
            _service.login(_sic.getSessionManager(), userName, password, null);
            ManagedObjectReference extMgrMof = _sic.getExtensionManager();
            _service.registerExtension(extMgrMof, ext);
        }
    }
    public static void main(String [] args) throws Exception {
        RegMyPlugin obj = new RegMyPlugin();
        String serverName = args[0];
        String userName = args[1];
        String password = args[2];
        obj.connLoginReg(serverName, userName, password);
    }
} //RegMyPlugin class

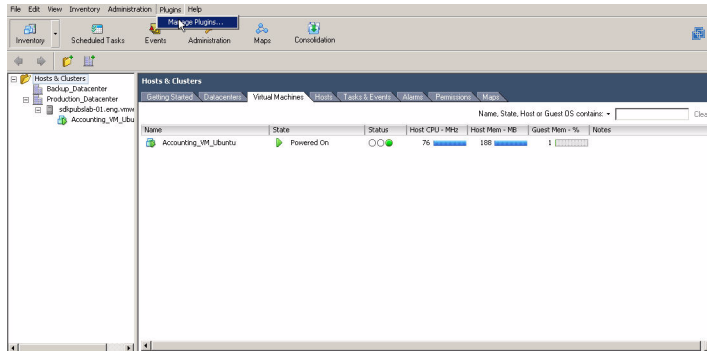
```

---

## Managing VC Plug-ins

After a VC Plug-in has been registered with the VirtualCenter Server ExtensionManager, the extended functionality will be available to users who connect to the VirtualCenter Server (see “VirtualCenter Server ExtensionManager” on page 3). VI Client users can enable or disable a specific plug-in by using the Manage Plug-ins... menu selection in VI Client (see Figure 6).

**Figure 6.** Managing VC Plug-ins in the VI Client



Note that the RegisterExtension method actually “registers” only the URL to a configuration file, not the configuration file itself. That means that if you move the configuration to a new location after registration, you will need to update the location or make other changes using other methods (operations) on the ExtensionManager API.

For example, to remove a VC Plug-in completely, use the UnregisterExtension operation. To change the location of its configuration file URL, use the UpdateExtension operation.

## Best Practices

Although this feature is considered experimental at this time, here are some best practices to consider when creating VC Plug-ins:

- **Keep it simple.** When planning where to add components to the VI Client, be sure to take into account the look and feel of the VI Client. Use similar fonts, font sizes, and color schemes for icons and other graphical user interface (GUI) components. Don't overload the Toolbar with too many icons.
- **Use the right tools.** The VI Client currently does not provide any debugging information, so always use an XML validation tool, such as Altova XML Spy, to validate the configuration file before posting it to the Web server. See <http://www.altova.com> for information about XML Spy.
- **Keep it secure.** Use HTTPS (rather than plain HTTP). By default, the VirtualCenter Server uses SSL. You should run the Web applications that you want to integrate with VirtualCenter on HTTPS as well, rather than HTTP.
- **Use unique keys.** Configuration keys must be unique across the VirtualCenter Server. VMware recommends using Java package naming conventions to ensure uniqueness: for example, com.yourCompanyName.ProductName.
- **Keep it stable.** Change data on the back-end carefully. Remember that if you make major changes to your Web server application, you may need to update the URL registration.