VMRC SDK Programming Guide

vSphere 5.5

vCloud Director 5.5

This document supports the version of each product listed and supports all subsequent versions until the document is replaced by a new edition. To check for more recent editions of this document, see http://www.vmware.com/support/pubs.

EN-001221-02

vmware®

You can find the most up-to-date technical documentation on the VMware Web site at:

http://www.vmware.com/support/

The VMware Web site also provides the latest product updates.

If you have comments about this documentation, submit your feedback to:

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

About This Book

The *VMRC SDK Programming Guide* provides information about developing applications by using the VMware Remote Console software development kit that is included with vSphere and vCloud Director.

VMware provides many different APIs and SDKs for various applications and goals. This book provides information about using the VMware Remote Console API for developers that are interested in creating web applications that can remotely access virtual machine console and device functions.

To view the current version of this book as well as all VMware API and SDK documentation, go to http://www.vmware.com/support/pubs/sdk_pubs.html.

Revision History

This book is revised with each release of the product or when necessary. A revised version can contain minor or major changes. Table 1 summarizes the significant changes in each version of this book.

Table 1. Revision History

Revision Date	Description
14 February 2014	Added index in December 2013. Later updated OS and browser compatibility information.
19 September 2013	Release of VMRC SDK for vSphere 5.5 and vCloud Director 5.5. Last revised 30 August.
10 September 2012	Initial release of VMRC SDK for vSphere 5.1 and vCloud Director 5.1.
12 October 2012	Documented differences in using getVirtualDevices() and getPhysicalClientDevices() when using VMRC with the Internet Explorer browser.

Intended Audience

This book is intended for anyone who needs to develop applications using the VMRC SDK. Typically this includes software developers who are creating Web applications using JavaScript, and who are targeting virtual machine remote-console functions in vSphere and vCloud Director.

VMware Technical Publications Glossary

VMware Technical Publications provides a glossary of terms that might be unfamiliar to you. For definitions of terms as they are used in VMware technical documentation go to http://www.vmware.com/support/pubs.

Document Feedback

VMware welcomes your suggestions for improving our documentation. Send your feedback to docfeedback@vmware.com.

Remote Console Access with VMRC 1

This chapter contains the following topics:

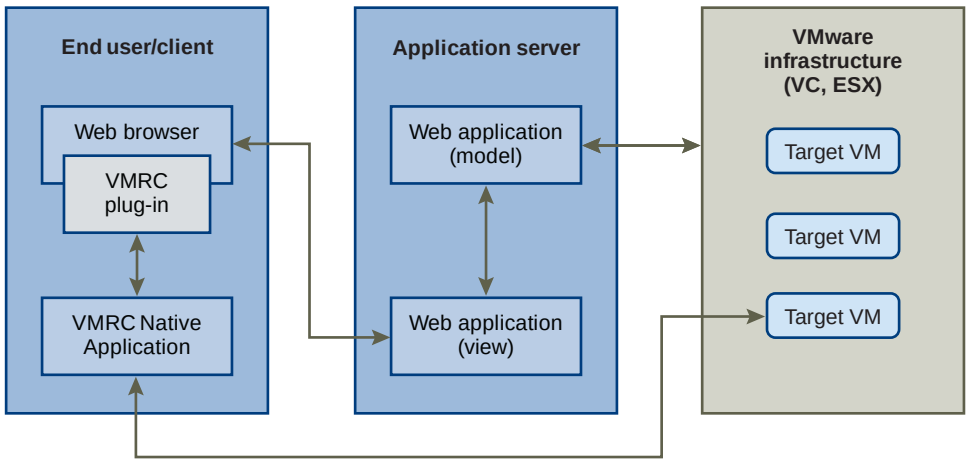
With vSphere and vCloud Director, you can create Web-based applications that remotely access virtual machines and perform console and device operations. Both vSphere and vCloud Director incorporate the VMware Remote Console browser plug-in, which can be loaded in supported Web browsers. Applications running in the browser can use the VMRC browser plug-in to access virtual machine console functions through a JavaScript API. With applications that use the VMRC browser plug-in and VMRC API, users can remotely interact with virtual machines from any supported operating system and Web browser.

Overview of VMRC Architecture

To use VMRC, a client system uses a Web browser-based application to load the VMRC browser plug-in. The Web application can then use the VMRC JavaScript API to issue console and device commands by calling the API methods. When used with vSphere, the target virtual machine or vCenter Server acts as the server side of the client-server relationship. Figure 1-1 shows the VMRC architecture when used with vSphere.

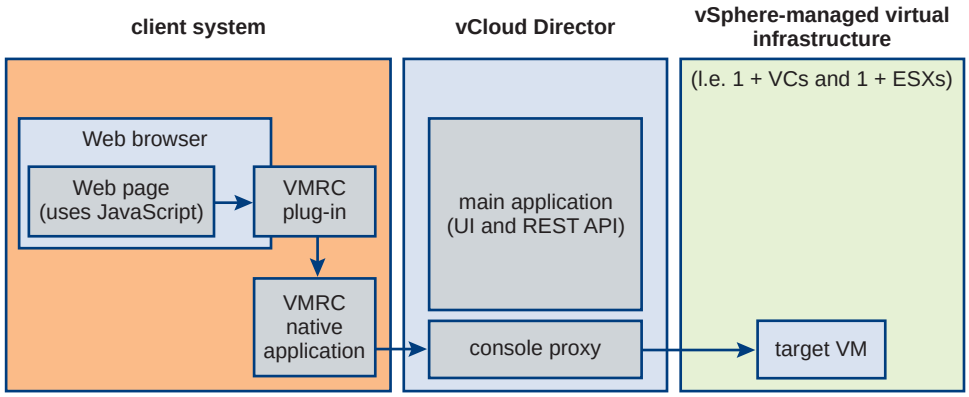
Your Web application can use the VMRC API methods to connect to a virtual machine and perform remote operations. The VMRC API contains methods for console commands, such as changing the virtual machine's screen mode or sending a Control-Alt-Delete key sequence to the virtual machine. The VMRC API also contains methods for connecting physical devices on the client to virtual machines.

Figure 1-1. VMRC Architecture with vSphere



When used with vCloud Director, the client communicates with the target virtual machine through a vCloud Director console proxy. Figure 1-2 shows the VMRC architecture when used with vCloud Director.

Figure 1-2. VMRC Architecture with vCloud Director



The VMRC API contains callback signals that are sent by the browser plug-in when there are changes to the virtual machine's state. Your Web application can handle these callback signals to respond to events on the virtual machine. Most VMRC callback signals are closely associated with VMRC API methods and serve as feedback when your web application calls a method. For example, a successful call to the `connect()` API method causes the VMRC browser plug-in to generate an `onConnectionStateChange` event, while a call to the `setFullscreen()` method results in an `onFullscreenChange` event.

Requirements for Using the VMRC Browser Plug-In

To use VMRC, your Web application must be able to load the VMRC browser plug-in. The VMRC browser plug-in is supported for use with Microsoft Windows XP or later, and the Linux operating system. The VMRC browser plug-in cannot be easily installed and is not supported with Mac OS X.

VMRC is supported by all vSphere and ESXi host configurations, and by all vCloud Director configurations. Your Web application must be written in JavaScript to make use of the VMRC API.

The Windows version of the VMRC browser plug-in is available only as a 32-bit application. The Linux version of the VMRC browser plug-in is available as a 32-bit or a 64-bit application. The VMRC browser plug-in for vSphere and vCloud Director was originally tested with the following web browsers.

- Microsoft Internet Explorer version 7 or later
- Mozilla Firefox version 9 or later

The following browser is supported when using VMRC with vSphere but not with vCloud Director.

- Google Chrome version 16 or later

VMRC currently calls the Netscape Plug-in API (NPAPI), which will eventually be removed from Chrome.

VMRC Workflow

To use VMRC, the VMRC browser plug-in must be installed on the client system. A Web application using VMRC typically must perform the following actions.

1. Load the VMRC browser plug-in using the appropriate JavaScript call and constant value.
2. Set JavaScript event handlers to respond to VMRC callback signals.
3. Start the VMRC plug-in using the `startup()` API method.
4. Use the VMRC API `connect()` method to connect to a target virtual machine.
5. Use the VMRC API methods to send commands to the target virtual machine.
6. Use the VMRC API `disconnect()` method to disconnect from the target virtual machine.
7. Shut down the VMRC browser plug-in using the `shutdown()` API method.

Setting Up the VMRC Browser Plug-In 2

This chapter contains the following topics:

To use VMRC, your Web application must first load the VMRC browser plug-in, and also must set up the property value constants and callback handler methods required by the VMRC API.

Once your Web application has performed these initial steps, you can use the VMRC API methods to connect to a virtual machine and access console and device functions.

Using VMRC Property Value Constants

To use the VMRC API with the VMRC browser plug-in, you must set up the necessary property value constants that the API methods require. In your web Application JavaScript code, you must use these property value constants when invoking VMRC API methods or handling callback signals.

The syntax of the property value constants vary depending on which web browser you use with VMRC browser plug-in.

In Internet Explorer, the property value constants are exposed as dictionaries corresponding to each property value class name. For example, for the `VMRC_ConnectionState` variable, a value of `VMRC_CS_CONNECTED` is represented by the following JavaScript code in Internet Explorer:

```
vmrc.VMRC_ConnectionState("VMRC_CS_CONNECTED")
```

In Firefox or Chrome, the property value constants directly correspond to the property name. For the `VMRC_ConnectionState` variable, a value of `VMRC_CS_CONNECTED` is represented by the following JavaScript code in Firefox or Chrome:

```
vmrc.VMRC_ConnectionState.VMRC_CS_CONNECTED
```

The VMRC SDK contains example JavaScript code that abstracts these differences in handling property value constants. When developing your Web application, you can use the example code to ensure that the VMRC property value constants are accessed in a uniform manner, regardless of which Web browser the user runs.

Loading the VMRC Browser Plug-In

Your Web application must load the VMRC browser plug-in by creating an object instance of the plug-in. If you create this object instance for Internet Explorer browsers, you must set the instance class ID to a value specific to the version of the VMRC API you want to use. If you create the object instance in the Firefox or Chrome browsers, you must likewise set the object instance `type` attribute to a specific value for your VMRC version. Example 2-1 shows how to create an object instance using HTML, and how to set the `CLSID` or `type` values. The object instance is assigned the ID value `vmrc`.

Example 2-1. Creating an Object Instance of the VMRC Browser Plug-In

```
<!--[if IE]>
<object id="vmrc" classid="CLSID:4AEA1010-0A0C-405E-9B74-767FC8A998CB"
                style="width: 100%; height: 100%;"></object>
<![endif] -->
<!--[if !IE]><!-->
<object id="vmrc" type="application/x-vmware-remote-console-2012"
                style="width: 100%; height: 100%;"></object>
<!--<![endif]-->
```

The code in Example 2-1 can be included anywhere in the HTML code that makes up your Web application. Subsequent VMRC API examples in this document refer to the plug-in object instance using the value `vmrc` as set in the example.

The values of the `classid` and `type` attributes correspond to specific versions of the VMRC API. For the vSphere 5.1 and vCloud Director 5.1 releases, the `classid` must be set to the following value.

```
CLSID:4AEA1010-0A0C-405E-9B74-767FC8A998CB
```

In vSphere 5.1 and vCloud Director 5.1, the `type` attribute must be set to the following value.

```
application/x-vmware-remote-console-2012
```

You can verify the VMRC browser plug-in version prior to starting the VMRC plug-in by using the `getVersion()` API method. See Chapter 3, "Using the VMRC Plug-In and the API," on page 13.

Windows Plug-in

In the SDK, the `VMware-ClientIntegrationPlugin-5.5.0.exe` installer applies to Windows systems.

VMware provides only the 32-bit binary VMRC plug-in, which also runs on 64-bit Windows.

Linux 32-Bit and 64-Bit

The `VMware-ClientIntegrationPlugin-5.5.0.i386.bundle` installer applies to 32-bit Linux, and the `VMware-ClientIntegrationPlugin-5.5.0.x86_64.bundle` installer applies to 64-bit Linux.

You must be logged in as root or have superuser privileges to run the bundle installers. The plug-in will be installed in `/usr/lib/mozilla/plugins` as shown below.

```
# ls -l /usr/lib/mozilla/plugins
...
lrwxrwxrwx 1 root root 62 Apr 8 14:58 npVMwareClientSupportPlugin-5-5-0.so ->
               /usr/lib/vmware-cip/5.5.0/npVMwareClientSupportPlugin-5-5-0.so
lrwxrwxrwx 1 root root 59 Apr 8 14:58 np-vmware-vmrc-5.5.0-1071230.so ->
               /usr/lib/vmware-vmrc/5.5/np-vmware-vmrc-5.5.0-1071230-64.so
```

If you run 32-bit Firefox on 64-bit Linux, install just the `i386` bundle.

If you run 64-bit Firefox on 64-bit Linux, The plug-in directory might be `/usr/lib64/mozilla/plugins` instead of `/usr/lib/mozilla/plugins`. If the plug-in directory does not exist, you must create it.

If you are not certain whether you have 32-bit or 64-bit Firefox installed, first try `/usr/lib`, and if that does not work, try `/usr/lib64`. If you compiled your own copy of Firefox, similar advice applies. If the compiled Firefox binary is under your home directory as `firefox/firefox`, the plug-in should be installed in the `firefox/plugins` subdirectory.

Setting Handlers for VMRC Events

When there is a change in the state of a VMRC session, the VMRC browser plug-in generates events using the VMRC API. Events generated include changes in connection state, changes in screen size on the target virtual machine, or events generated in response to messages from the VMRC plug-in. See Chapter 4, "Handling VMRC Events," on page 25.

To use the VMRC browser plug-in, your Web application must set up handlers for the VMRC events. You must bind the events to JavaScript handler methods in your Web application. The VMRC API provides different binding mechanisms for each supported Web browser.

Setting Handlers in Internet Explorer

In Internet Explorer, handler methods are bound using the `attachEvent()` method. You set the handler method by calling `attachEvent()` and passing as parameters the event name as a string, and a pointer to the JavaScript handler method.

For example, to set a handler method for the `onConnectionStateChange` event, you must call `attachEvent()` as follows:

```
vmrc.attachEvent("onConnectionStateChange", onConnectionStateChangeHandler);
```

The first parameter to `attachEvent()` is the event name that is defined in the VMRC API. The second parameter is the JavaScript handler method that you define.

Setting Handlers in Firefox or Chrome

In Firefox and Chrome, each event corresponds to a property of the VMRC plug-in object instance. You bind a handler method to a VMRC event by setting the value of the corresponding event property to the handler method.

For example, to set a handler method for the `onConnectionStateChange` event, you must set the VMRC object instance's `onConnectionStateChange` property as follows:

```
vmrc["onConnectionStateChange"] = onConnectionStateChangeHandler;
```

Abstracting Event Handler Setup

The VMRC SDK contains sample code that shows how to abstract the setting of event handlers into a single function that can be used with all supported browsers. See the `attachEventHandler()` method in the sample file `vmrc-embed-example.js` in the VMRC SDK binaries.

Accessing VMRC Log Files

Some components in the VMRC SDK automatically produce log files. You can use these log files for tracing VMRC behavior for debugging and optimization. When reporting bugs or other issues with VMRC, as a best practice, include the locations of the directories containing the relevant log files.

The VMRC components automatically generate log files.

- VMRC Application Log. The application log is a log file produced by the client VMRC binary that processes VMRC API calls.
- VMRC Plug-In Log. The plug-in log file produced by the VMRC browser plug-in library that client Web applications use to call API methods.
- Remote MKS Log. The mouse keyboard screen (MKS) log is produced by an MKS binary on the local client machine. The MKS binary provides display data and metadata used to show the target virtual machine's console.
- USB Arbitrator Log. The USB arbitrator log is produced by the privileged USB service that provides the data and functionality used to connect local client USB devices to remote virtual machines.

Windows Log File Locations

Table 2-1 shows the location of each log file on the Windows operating system.

Table 2-1. Windows Log File Locations

Log File	Location
VMRC Application Log	`%TEMP%\vmware-%USERNAME%\`
VMRC Plug-In Log	`%TEMP%\vmrc-plugin\`
Remote MKS Log	`%TEMP%\vmware-%USERNAME%\`
USB Arbitrator Log	`%SYSTEMROOT%\Temp\vmware-SYSTEM\`

Linux Log File Locations

The VMRC browser plug-in does not generate a log file on the Linux operating system. You can access plug-in log information by running your Web browser from the command line. The VMRC browser plug-in generates logging information on `stdout`.

You can find logs generated by the VMRC native application process, as well as a Remote MKS process log, at the following location.

```
/tmp/vmware-$USER/
```

You can find the USB arbitrator log file at the following location.

```
/tmp/vmware-root/
```

Using the VMRC Plug-In and the API 3

This chapter contains the following topics:

After your Web application has loaded and otherwise prepared the VMRC browser plug-in, you can use the VMRC API to connect to a virtual machine and perform operations. Depending on which mode you choose when you start the VMRC browser plug-in process, you can use VMRC API methods to interact with the virtual machine's user interface (mouse, keyboard, and screen) or the virtual machine's devices. The VMRC API also contains general-purpose methods that you can use regardless of the VMRC browser plug-in mode.

To use the VMRC API, your Web application must typically perform the following steps.

1 Initialize and start the VMRC plug-in.

2 Connect to the remote host of the target virtual machine.

3 Call methods that correspond to the desired console or device commands on the target virtual machine.

4 Disconnect from the remote host.

5 Shut down the VMRC plug-in.

The VMRC API methods generate exceptions if a failure or an error occurs when you call a method. It is a best practice to use `try` and `catch` blocks in your web application code to handle any exceptions generated by the VMRC API methods. As many VMRC API methods do not return a value, handling the exceptions is the only way to handle errors and failures.

Starting the VMRC Browser Plug-In

Before you call any other API methods, your Web application must first initialize the VMRC browser plug-in. You can also use VMRC API methods to verify the VMRC browser plug-in version before starting the plug-in.

Verifying the VMRC Browser Plug-In Version

You can use the `getVersion()` API method to obtain the specific version of the VMRC browser plug-in and VMRC SDK that is installed on the local client system. You can call `getVersion()` at any time after loading the VMRC plug-in, even before you start the plug-in.

The `getVersion()` method is present in every version of the VMRC API. For detailed information on calling `getVersion()`, see "General-Purpose API Methods" on page 18.

The specific VMRC API version that your Web application uses depends on the `CLSID` or MIME type that your application supplied when loading the VMRC browser plug-in. See "Loading the VMRC Browser Plug-In" on page 9.

Initializing the VMRC Browser Plug-In

To initialize the VMRC browser plug-in, you must use call the following API methods in sequence.

1 Use the `isReadyToStart()` API method to determine whether the VMRC browser plug-in has been successfully loaded and is ready to be started.

2 Use the `startup()` API method to start the processes in the VMRC browser plug-in.

Using isReadyToStart()

The `isReadyToStart()` method takes no parameters and returns a boolean value. A return value of `true` indicates that the VMRC browser plug-in has been loaded and is ready to start. You may call `startup()` once the `isReadyToStart()` method returns a value of `true`.

NOTE The VMRC browser plug-in does not generate an event when the plug-in is ready to start. You must poll the return value of the `isReadyToStart()` method to determine whether the VMRC browser plug-in has loaded and can be started.

An example call to `isReadyToStart()` might appear as follows:

```
var ret = vmrc.isReadyToStart();
```

Using the startup() Method

You must use the `startup()` API method to start the processes in the VMRC browser plug-in. When you call the startup() method, you also set the modes in which the VMRC browser plug-in can operate. The available modes are Mouse-Keyboard-Screen (MKS) mode and Devices mode.

You can choose to start the VMRC browser plug-in with one or both of the available mode settings. The `mode` parameter to the `startup()` API method accepts a mask of values. You can include one or both mode settings in the value that you pass for the `mode` parameter by using the bitwise `or` (`|`) operator.

MKS Mode

When you start the VMRC browser plug-in using MKS mode, the screen contents of the target virtual machine are displayed in the local browser window. The user can interact with the input to the target virtual machine, such as the mouse and keyboard. The VMRC API contains methods specific to MKS mode, where your Web application can interact with the screen, send key sequences, or grab the virtual machine input. See "MKS Mode API Methods" on page 18.

You start the VMRC browser plug-in using MKS mode by calling the `startup()` method and including the property value constant `VMRC_Mode.VMRC_MKS` in the `mode` parameter.

Devices Mode

When you start the VMRC browser plug-in using Devices mode, you can use VMRC to manage virtual devices. You can connect virtual devices on the target virtual machine to physical devices on the local system. The VMRC API contains methods specific to Devices mode, where your Web application can connect physical client devices to virtual devices, and allow the target virtual machine to access physical devices on the local system that is running your Web application. See "Devices Mode API Methods" on page 20.

You start the VMRC browser plug-in using Devices mode by calling the `startup()` method and including the property value constant `VMRC_Mode.VMRC_DEVICES` in the `mode` parameter.

Table 3-1. Parameter List for VMRC startup() Method

Parameter Name	Type	Description/Notes
`mode`	`VMRC_Mode` property value constant; one or both of the values `VMRC_Mode.VMRC_MKS` and `VMRC_Mode.VMRC_DEVICES` (both can be conjoined using the | operator)	The `mode` parameter is specified at startup, and determines the operational mode of the VMRC plug-in instance for the lifetime of its connection. The plug-in can operate in the following modes. **VMRC_Mode.VMRC_MKS:** In MKS mode, guest screen contents are displayed and the user can interact with the guest. **VMRC_Mode.VMRC_DEVICES**: In Devices mode, the user can manage remote device backings and their mappings to virtual machine devices.
`msgmode`	`VMRC_MessageMode` property value constant; `VMRC_MessageMode.VMRC_EVENT_MESSAGES`, `VMRC_MessageMode.VMRC_DIALOG_MESSAGES`, `VMRC_MessageMode.VMRC_NO_MESSAGES`	The `msgmode` parameter is specified at startup, and determines by which mode messages are delivered from the plug-in to the containing document element in your Web application. The `VMRC_MessageMode.VMRC_DIALOG_MESSAGE` and `VMRC_MessageMode.VMRC_NO_MESSAGES` values are not supported on the Linux operating system.
`advancedconfig`	string	A string containing advanced configuration options. Values for this parameter must follow the format "flag=value", as in "usebrowserproxy=true". See Table 3-2 for valid configuration flags.

The `startup()` method returns a string value, which is implementation-specific and opaque to the client.

Advanced Configuration Flags

Table 3-2 shows the available advanced configuration flags that you can use in the `startup()` method's `advancedconfig` parameter. To use a flag, you must include the string "`{flag}=true`" in the `advancedconfig` parameter, where `{flag}` is your flag of choice. You can include multiple flags by separating each flag and value with a semicolon.

When using VMRC with vCloud Director, you must include advanced configuration flags `usebrowserproxy` and `tunnelmks`, with the values set to `true`.

Table 3-2. Advanced Configuration Flags for VMRC startup() Method

Flag	Description
`usebrowserproxy`	If true, VMRC connections use the same proxy settings as the Web browser running the VMRC browser plug-in.
`autoanswerquestions`	If true, end-user questions from the connected Virtual Machine are automatically answered by VMRC using the default choice that the Virtual Machine designates. You can use the `autoanswerquestions` flag for simple or devices-only clients that do not require end-user interaction.
`cacheconnections`	If true, VMRC preserves device connections to a given Virtual Machine even after VMRC disconnects from that Virtual Machine using the `disconnect()` method. You can use the `cacheconnections` flag for advanced clients that provide client device functionality for multiple Virtual Machines with a single VMRC instance.
`tunnelmks`	If true, VMRC MKS console traffic is tunneled over HTTPS.

Example Call

An example call to `startup()` in the Firefox browser might appear as follows:

```
var ret = vmrc.startup(vmrc.VMRC_Mode.VMRC_MKS, vmrc.VMRC_MessageMode.VMRC_EVENT_MESSAGES,
                "usebrowserproxy=true;tunnelmks=true");
```

Connecting to a Remote Host and Virtual Machine

After your Web application has initialized the VMRC browser plug-in, you can use the `connect()` method to connect to a remote host and access a particular virtual machine on that host. To use the `connect()` method, you must have the following information.

- The `hostname` or IP address of the remote host and the `thumbprint` of the host SSL certificate.
- A form of authentication, which can be either a `username` and `password` pair, or a VIM session ticket.
- A form of identification for the target virtual machine, which can be either a virtual machine ID or a `datacenter` and `vmpath` pair.

You pass different parameters to the `connect()` method depending on how you choose to provide the required host identification, authentication, and virtual machine identification. Table 3-3 shows the parameters for the `connect()` method.

Table 3-3. Parameter List for VMRC connect() Method

Parameter Name	Type	Description/Notes
`host`	string	Hostname or IP address of the remote host running the target virtual machine.
`sslThumbprint`	string	Expected thumbprint of the remote host's SSL certificate. You must provide the SSL thumbprint if you authenticate the connection by using a username and password pair.
`allowSSLErrors`	boolean	Boolean value that determines whether or not the host SSL certificate validation checker allows connections that contain validation errors. You must provide a value in the `sslThumpbrint` parameter if you set the `allowSSLErrors` parameter to `true`.
`ticket`	string	VIM session ticket, used to authenticate with the remote host. You obtain the VIM session ticket by using the VIM call `acquireCloneTicket()` on an active VIM session. If you use a ticket to authenticate, you must not pass a value for the `username` or `password` parameters. The `sslThumbprint` and `allowSSLerrors` parameters are optional if you authenticate using the ticket.
`username`	string	Username to authenticate with the remote host. If you pass a username and password pair to authenticate, do not pass a value for the `ticket` parameter. You must also provide an SSL thumbprint using the `sslThumbprint` parameter.
`password`	string	Password, used to authenticate with the remote host.
`vmid`	string	Virtual machine ID, used to identify the target virtual machine. If you use the `vmid` parameter to specify the target virtual machine, you must not pass a value for the `datacenter` or `vmpath` parameters.
`datacenter`	string	Datacenter name on which the target virtual machine is stored, used to identify the target virtual machine. If you use a datacenter and vmpath pair to identify the target virtual machine, you must not pass a value for the `vmid` parameter.
`vmpath`	string	Full datastore path to the target virtual machine, of the format [datastore]vmpath(vm.vmx).

The `connect()` method does not return a value. If the connection is successful, the VMRC browser plug-in generates an `onConnectionStateChange` event. See Chapter 4, "Handling VMRC Events," on page 25.

NOTE When using VMRC with vCloud Director, only the `host`, `ticket`, and `vmid` parameters are supported when calling `connect()`. You must pass `false` for the `allowSSLErrors` parameter, and empty strings ("") for all other parameters in `connect()`.

The following is an example of how to call `connect()` when using VMRC with vCloud Director.

```
connect(hostname, "", false, ticket, "", "", vmid, "", "");
```

Connection Authentication

You must provide a form of authentication when using the `connect()` method. This authentication can be a VIM session ticket, or a `username` and `password` pair on the remote host. The authentication methods are exclusive, meaning that if you use the parameters for one method, such as a session ticket, you must omit the parameters for the other and pass them as empty strings.

The following is an example of how to call `connect()`, by using a VIM session ticket as the authentication method. Note that the `username`, `password`, and SSL `thumbprint` parameters are passed as empty strings.

```
connect(hostname, "", false, ticket, "", "", vmid, "", "");
```

If you authenticate using a `username` and `password` pair, you must provide the SSL `thumbprint` to the `connect()` method in the `sslThumbprint` parameter.

NOTE Username and password authentication is not supported when using VMRC with vCloud Director. You must use the `ticket` parameter when calling `connect()` in a vCloud Director configuration.

The following is an example of how to call `connect()`, by using a `username` and `password` as the authentication method. The ticket parameter is passed as an empty string.

```
connect(hostname, sslThumb, false, "", uname, pass, vmid, "", "");
```

Virtual Machine Identifier

You must provide a way to identify the target virtual machine when using the `connect()` method. The virtual machine can be identified by a virtual machine ID that you obtain from an active VIM session on your Web application server, or a datacenter name and datastore path to a particular virtual machine.

The two methods of identifying the virtual machine are exclusive. If you use a virtual machine ID, you must pass that value by using the `vmid` parameter when calling `connect()`, and omit any values for `datacenter` and `vmpath`. Conversely, you must omit the `vmid` if you use values for `datacenter` and `vmpath` when calling `connect()`.

NOTE If you use VMRC with vCloud Director, you must use the virtual machine ID to identify the target virtual machine when calling `connect()`.

The following example shows how to call `connect()` by using the virtual machine ID to identify the target virtual machine.

```
connect(hostname, "", false, ticket, "", "", vmid, "", "");
```

The following is an example of how to call `connect()` using the datacenter name and vmpath to identify the target virtual machine.

```
connect(hostname, "", false, ticket, "", "", "", dstore, vmpath);
```

Calling VMRC API Methods on a Virtual Machine

After you have established a connection to the virtual machine remote host, you can use the VMRC API methods to interact with the target virtual machine.

Most methods in the VMRC API are associated with a specific mode of the VMRC browser plug-in, such as MKS mode or Devices mode. Methods associated with a particular operational mode are only available when the VMRC browser plug-in is started in that mode. For example, the MKS mode method `setFullscreen()` is only available if you specified `VMRC_Mode.VMRC_MKS` when starting up the VMRC browser plug-in. See "Starting the VMRC Browser Plug-In" on page 13.

Some methods in the VMRC API pertain only to the VMRC browser plug-in and can be used at any time, including before `startup()` has been called. These methods generally provide version information on VMRC and information about the supported APIs.

General-Purpose API Methods

General-purpose API methods provide information about VMRC and the APIs it supports. These methods can be called at any time, including before you call the `startup()` method for the VMRC browser plug-in.

getVersion()

The `getVersion()` method retrieves the current complete version number of the installed VMRC browser plug-in.

Method	`getVersion()`
Parameters	None
Return Value	`string`; contains the full version number for the VMRC plug-in
Example Call	`var version = vmrc.getVersion();`

getConnectionState()

The `getConnectionState()` method retrieves the current connection state from a VMRC browser plug-in.

Method	`getConnectionState()`
Parameters	None
Return Value	Property Value Constant; valid values are `VMRC_CS_CONNECTED` or `VMRC_CS_DISCONNECTED`
Example Call	`var ret = vmrc.getConnectionState();`

MKS Mode API Methods

When you use the VMRC browser plug-in using MKS mode, the VMRC processes provide access to the target virtual machine console. VMRC connects to the display console of the remote virtual machine and that display appears in your Web application window. When you use the VMRC browser plug-in using MKS mode, you can use the following VMRC API methods.

setScreenSize()

The `setScreenSize()` method commands the VMRC browser plug-in to set the screen resolution of the currently connected virtual machine.

Method	`setScreenSize(width, height)`
Parameters	width (`int`); represents the desired screen width of the console, in pixels height (`int`); represents the desired screen height of the console, in pixels
Return Value	`void`
Example Call	`vmrc.setScreenSize(w, h);`

If the call to `setScreenSize()` is successful, the VMRC browser plug-in generates an `onScreenSizeChange()` event. If the call is unsuccessful, an exception is thrown.

screenWidth()

The `screenWidth()` method retrieves the screen width, in pixels, of the currently connected virtual machine.

Method	`screenWidth()`
Parameters	None
Return Value	`int`; represents screen width in pixels
Example Call	`var sw = vmrc.screenWidth();`

screenHeight()

The `screenHeight()` method retrieves the screen height, in pixels, of the currently connected virtual machine.

Method	`screenHeight()`
Parameters	None
Return Value	`int`; represents screen height in pixels
Example Call	`var sh = vmrc.screenHeight();`

setFullscreen()

The `setFullscreen()` method commands the VMRC browser plug-in to enter or exit full-screen mode.

Method	`setFullscreen(fs)`
Parameters	`fs (boolean)`; set to `true` to enter full-screen mode, `false` to exit
Return Value	`boolean`; `true` for success or `false` for failure
Example Call	`var ret = vmrc.setFullscreen(true);`

If the call to `setFullscreen()` is successful, the VMRC browser plug-in generates an `onFullscreenChange()` event. If the call is unsuccessful, an exception is thrown.

getFullScreen()

The `getFullscreen()` method retrieves the current state of the full-screen mode of the VMRC browser plug-in.

Method	`getFullscreen()`
Parameters	None
Return Value	`boolean`; `true` if full-screen mode is set, `false` if full-screen mode is not set
Example Call	`var ret = vmrc.getFullscreen();`

sendCAD()

The `sendCAD()` method sends a Control-Alt-Delete key sequence to the currently connected virtual machine.

Method	`sendCAD()`
Parameters	None
Return Value	`boolean`; `true` for success or `false` for failure
Example Call	`var ret = vmrc.sendCAD();`

grabInput()

The `grabInput()` method commands the VMRC browser plug-in to "grab" or capture the current mouse and keyboard input and send it to the currently connected virtual machine console.

Method	`grabInput()`
Parameters	None
Return Value	`void`
Example Call	`vmrc.grabInput();`

If the call to `grabInput()` is successful, the VMRC browser plug-in generates an `onGrabStateChange()` event. If the call is unsuccessful, an exception is thrown.

ungrabInput()

The `ungrabInput()` method commands the VMRC browser plug-in to discontinue capturing the current mouse and keyboard input.

Method	`ungrabInput()`
Parameters	None
Return Value	`void`
Example Call	`vmrc.ungrabInput();`

If the call to `ungrabInput()` is successful, the VMRC browser plug-in generates an `onGrabStateChange()` event. If the call is unsuccessful, an exception is thrown.

setInputRelease()

The `setInputRelease()` method commands the VMRC browser plug-in to disable all mouse and keyboard input capture, regardless of the current grab state. You can use this method to lock any input from reaching the connected VM. For example, if your web application displays a modal dialog, or enters a state in which you want to prevent any input from reaching the connected VM, you can call `setInputRelease()` with a parameter value of `true`.

You can call `setInputRelease()` with a parameter value of `false` to revert the VMRC browser plug-in to normal input behavior. When you have done so, you can change the input grab state using the `grabInput()` and `ungrabInput()` methods.

Method	`setInputRelease(release)`
Parameters	`release (boolean)`; set to `true` to lock input capture, `false` to revert to normal grab/ungrab behavior
Return Value	`void`
Example Call	`vmrc.setInputRelease(true);`

Devices Mode API Methods

When you use the VMRC browser plug-in using Devices mode, you can use the physical devices on the local client machine with the currently connected virtual machine. To use devices in this way, you must connect the local physical devices to remote virtual device backings by using methods in the VMRC API. The connection methods differ for USB and non-USB devices.

NOTE Devices mode API methods are not supported for use with USB devices when using VMRC with vCloud Director.

Obtaining Device Keys and Connecting Devices

To connect non-USB devices, such as a CD-ROM or floppy disk drive, you use the VMRC API to obtain both a physical and virtual device key. A device key is a unique identifier for a local physical device or a remote virtual device. You then use the device keys with the `connectDevice()` VMRC API method to connect the devices. For USB devices, only the physical device key for the local client device is required.

For file-backed physical devices, such as a file backing for a CD-ROM or floppy device, you must substitute the full path to the device file for the physical key.

To connect a local physical device to a remote virtual device

1 Use the `getPhysicalClientDevices()` API method to obtain a list of device keys for the physical devices on the local machine.

   The `getPhysicalClientDevices()` method returns a physical device key for each device that is available for remote connection.

2 Use the `getPhysicalClientDeviceDetails()` API method with the desired physical device key to obtain more information about the specific physical device you want to connect.

3 If you are connecting a non-USB device, use the `getVirtualDevices()` API method to obtain a list of device keys for the virtual devices on the currently connected virtual machine.

   The `getVirtualDevices()` method returns a virtual device key for each device that is available for remote connection. You do not provide a virtual device key when you connect a USB device.

4 If you are connecting a non-USB device, use the `getVirtualDeviceDetails()` API method with the appropriate virtual device key to obtain more information about the specific virtual device you want to connect, including the virtual device's remote backing type.

5 Use the `connectDevice()` API method to connect the local physical device to the corresponding remote virtual device.

   To use the `connectDevice()` method to connect a non-USB device, you must specify the device key for both the local physical device and remote virtual device, as well as the backing type. For USB devices, you must specify a blank parameter for the virtual device key, the physical device key for the local USB device, and the backing type.

   For file-backed physical devices, use the full path to the device file as the physical key.

6 When you are finished with device operations, use the `disconnectDevice()` API method to disconnect the device.

Managing USB Devices

To connect a USB device, you use the VMRC API to obtain a physical client device key for the local physical USB device. When you connect the local USB device to a remote virtual machine, the USB device becomes a new virtual USB device for that virtual machine. A connected USB device appears in both the local physical device list that you obtain using the `getPhysicalClientDevices()` method, and the remote virtual device list that you obtain using `getVirtualDevices()`.

NOTE USB devices are not supported when using VMRC with vCloud Director.

You can match the USB device on the local physical machine with the corresponding remote virtual device by performing the following steps.

To match a local physical USB device with a remote USB device connection

1 Obtain the virtual device key for the remote virtual USB device on the connected virtual machine.

   You can use the `getVirtualDevices()` method and specify `VMRC_DeviceType.VMRC_DEVICE_USB` to retrieve only USB devices.

2 To obtain device details for the remote virtual USB device, use the `getVirtualDeviceDetails()` method, passing the virtual device key.

   The device detail `connectedByMe` indicates whether the remote virtual USB device is connected from your local physical machine. If `connectedByMe` is `true`, the remote virtual USB device corresponds to a local physical USB device. The device detail `backingKey` contains the physical device key for the local physical USB device.

   If `connectedByMe` is `false`, a different VMRC client has connected a physical USB device to the virtual machine. The device detail `hostName` provides the name of the machine that owns the physical USB device.

3 To obtain device details for the local physical USB device, use the `getPhysicalClientDeviceDetails()` method, passing the `backingKey` that you obtained in step 2.

getPhysicalClientDevices()

You use the `getPhysicalClientDevices()` method to obtain a list of physical device keys. These physical device keys correspond to physical client devices on the local machine, accessing your Web application, that are eligible for remove device connections. When you call the `getPhysicalClientDevices()` method, you pass a property value constant that specifies the types of devices for which to obtain device keys.

NOTE When using the `getPhysicalClientDevices()` method with the Internet Explorer browser, you must wrap the return value array as a `VBArray`. The following example shows how to process the return value of `getPhysicalClientDevices()` as a `VBArray`.

```
var devices = new VBArray(vmrc.getPhysicalClientDevices(mask)).toArray();
```

Method	`getPhysicalClientDevices(mask)`
Parameters	`mask (property value constant)`; the mask value specifies what types of device keys the method returns. The `mask` parameter can contain any combination of the values `VMRC_DEVICE_FLOPPY`, `VMRC_DEVICE_CDROM`, and `VMRC_DEVICE_USB`. You can also use `VMRC_DEVICE_ALL` to specify all device types. When using VMRC with vCloud Director, `VMRC_DEVICE_ALL` specifies only floppy and CD-ROM device types.
Return Value	`string[] vector` of physical device key strings.
Example Call	`var deviceKeys = vmrc.getPhysicalClientDevices(VMRC_DEVICE_FLOPPY | VMRC_DEVICE_CDROM);`

getPhysicalClientDeviceDetails()

You use the `getPhysicalClientDeviceDetails()` method to obtain detailed information about a particular local physical device. The `getPhysicalClientDeviceDetails()` method returns a JavaScript object that contains the device information.

Method	`getPhysicalClientDeviceDetails(physicalKey)`
Parameters	`physicalKey (string)`; the physical device key for the specified device, retrieved using the `getPhysicalClientDevices()` method.
Return Value	`JavaScript Object` that contains the following fields (keyed by strings): `key (string)`: Device key `type (VMRC_DeviceType)`: Device type `state (VMRC_DeviceState)`: Device state `connectedByMe (boolean)`: Whether the device is connected by the current VMRC instance `name (string)`: Device-friendly name `path (string)`: Device path `usbFamilies (VMRC_USBDeviceFamily)`: (USB only) Mask of VMRC_USBDeviceFamily property value constant values `usbSharable (boolean)`: (USB only) Whether the device is sharable by multiple VMRC instances `usbSpeeds (VMRC_USBDeviceSpeed)`: (USB only) Mask of VMRC_USBDeviceSpeed property value constant values
Example Call	`var ret = vmrc.getPhysicalClientDeviceDetails(key);`

getVirtualDevices()

You use the `getVirtualDevices()` method to obtain a list of virtual device keys. These virtual device keys correspond to virtual devices on the currently connected virtual machine that are eligible for remote device connections. When you call the `getVirtualDevices()` method, you pass a property value constant that specifies the types of devices for which to obtain device keys.

NOTE When using the `getVirtualDevices()` method with the Internet Explorer browser, you must wrap the return value array as a `VBArray`. The following example shows how to process the return value of `getVirtualDevices()` as a `VBArray`.

```
var devices = new VBArray(vmrc.getVirtualDevices(mask)).toArray();
```

Method	`getVirtualDevices(mask)`
Parameters	`mask (property value constant)`; the mask value specifies what types of device keys the method returns. The `mask` parameter can contain any combination of the values `VMRC_DEVICE_FLOPPY`, `VMRC_DEVICE_CDMROM`, and `VMRC_DEVICE_USB`. You can also use `VMRC_DEVICE_ALL` to specify all device types. When using VMRC with vCloud Director, `VMRC_DEVICE_ALL` specifies only floppy and CD-ROM device types.
Return Value	`string[] vector` of virtual device key strings.
Example Call	`var deviceKeys = vmrc.getVirtualDevices(VMRC_DEVICE_FLOPPY | VMRC_DEVICE_CDROM);`

getVirtualDeviceDetails()

You use the `getVirtualDeviceDetails()` method to obtain detailed information about a particular virtual device. The `getVirtualDeviceDetails()` method returns a JavaScript object that contains the device information.

Method	`getVirtualDeviceDetails(virtualKey)`
Parameters	`virtualKey (string)`; the virtual device key for the specified device, retrieved using the `getVirtualDevices()` method..
Return Value	`JavaScript Object` that contains the following fields (keyed by strings): `key (string)`: Device key `type (VMRC_DeviceType)`: Device type `state (VMRC_DeviceState)`: Device state `connectedByMe (boolean)`: Whether the device is connected by the current VMRC instance `name (string)`: Device-friendly name `hostName (string)`: Host name of the physical machine that provides the virtual device backing `clientBacking (boolean)`: Whether the device is configured to support a client backing `backingKey (string)`: Physical key for the physical device backing the virtual device `backing (VMRC_DeviceBacking)`: Device backing type
Example Call	`var ret = vmrc.getVirtualDeviceDetails(key);`

connectDevice()

You use the `connectDevice()` method to connect a physical client device to the currently connected virtual machine, either directly, for USB devices, or through a remote device backing, for non-USB devices. You must have an active connection to use the `connectDevice()` method and you must also have obtained the device keys for the specified physical client and remote virtual device for non-USB devices.

NOTE To connect a USB device, the remote virtual machine must have a USB controller installed. Connecting USB devices is not supported when using VMRC with vCloud Director.

Method	`connectDevice(virtualKey, physicalKey, backingType)`
Parameters	`virtualKey (string)`; The identifier of the target virtual machine's virtual device, which you can retrieve by using the `getVirtualDevices()` method. For USB devices, this parameter should be passed as an empty string (“”). `physicalKey (string)`; The identifier of the physical device on the client system, which you can retrieve by using the `getPhysicalClientDevices()` method. If you are using a physical device with a file backing, the `physicalKey` string must be the file path. `backingType (property value constant)`; This parameter indicates whether the `physicalKey` parameter refers to a physical device or a local file. Valid values include `VMRC_DB_FILE` and `VMRC_DB_PHYSICAL`. If you are using `VMRC_DB_FILE` to represent a CD-ROM or floppy device with a file backing, the physical key must be the absolute file path.

Return Value	`void`

Example Call	For non-USB devices (`pKey` refers to a physical CD-ROM device):
	`vmrc.connectDevice(vKey, pKey, VMRC_DB_PHYSICAL);`
	For USB devices (`pKey` refers to a physical USB device):
	`vmrc.connectDevice(“”, pKey, VMRC_DB_PHYSICAL);`
	For a file-backed physical device (`pKey` refers to a file path):
	`vmrc.connectDevice(vKey, pKey, VMRC_DB_FILE);`

If the call to `connectDevice()` is successful, the VMRC browser plug-in generates an `onDeviceStateChange()` event. If the call is unsuccessful, the VMRC browser plug-in throws an exception.

disconnectDevice()

You use the `disconnectDevice()` method to disconnect a physical client device from the currently connected virtual machine. You must have an active connection to use the `disconnectDevice()` method. For non-USB devices, you must specify the virtual device key of the remote virtual device to disconnect. For USB devices, you must specify the physical device key of the physical client device to disconnect.

Method	`disconnectDevice(deviceKey)`
Parameters	`deviceKey (string)`; The identifier of the device to disconnect. For non-USB devices, this must be the virtual key of the virtual device on the target virtual machine. For USB devices, this must be the physical key of the physical client device.
Return Value	`void`
Example Call	`vmrc.disconnectDevice(deviceKey);`

If the call to `disconnectDevice()` is successful, the VMRC browser plug-in generates an `onDeviceStateChange()` event. If the call is unsuccessful, the VMRC browser plug-in throws an exception.

Disconnecting and Shutting Down the VMRC Browser Plug-In

On termination, your Web application can clean up the VMRC processes by closing active connections and shutting down the VMRC browser plug-in.

Disconnecting an Active Connection

If your Web application has finished performing operations on the target virtual machine, or you otherwise want to close the connection, you can use the `disconnect()` API method to terminate the connection to the remote host. Disconnecting the VMRC browser plug-in from the target virtual machine terminates any active device connections. You can use an advanced configuration option to ensure that virtual device connections remain persistent after you call `disconnect()`.

The `disconnect()` method accepts no parameters and does not return a value. If a failure occurs, an exception is generated. If the call to `disconnect()` is successful, the VMRC browser plug-in will generate an `onConnectionStateChange()` event. An example call to the `disconnect()` method might appear as follows:

```
vmrc.disconnect();
```

Shutting down the VMRC Browser Plug-In

You can shut down the VMRC browser plug-in by invoking the `shutdown()` method. Shutting down the VMRC browser plug-in stops the corresponding VMRC peer processes within the plug-in, and terminates any active connections.

The `shutdown()` method accepts no parameters and does not return a value. An exception is generated if the call to `shutdown()` fails or encounters an error. An example call to the `shutdown()` method is as follows:

```
vmrc.shutdown();
```

Handling VMRC Events 4

This chapter contains the following topics:

The VMRC browser plug-in generates events in response to changes in the VMRC session, such as changes in the connection state or messages generated by the currently connected virtual machine. In your Web application, you can bind these VMRC events to JavaScript handler methods that are called when the VMRC browser plug-in generates the corresponding event. You set handler methods for the VMRC events to which you want to respond when you load and set up the VMRC browser plug-in. See Chapter 2, "Setting Up the VMRC Browser Plug-In," on page 9.

VMRC Event Parameters

Each event that the VMRC browser plug-in generates has associated parameters. The JavaScript handler method that you define for a particular event must accept the same set of parameters that the event API provides. For example, the VMRC event `onScreenSizeChange` provides two integer parameters: the new screen width and the new screen height. The handler method that you provide for the `onScreenSizeChange` event must likewise accept the two integer values for width and height as parameters.

For more information on defining handler methods as part of the VMRC browser plug-in setup process, see "Setting Handlers for VMRC Events" on page 11.

List of VMRC Events

The VMRC browser plug-in generates some events, such as message events and connection state change events, regardless of the mode you choose when starting the VMRC processes. Events related to changes in the console state, such as screen or input, are generated when the VMRC browser plug-in is started in MKS mode. Events related to changes in device state are generated only when the VMRC browser plug-in is started in Devices mode.

General-Purpose Events

The VMRC browser plug-in generates general-purpose events regardless of the mode you choose when starting the VMRC browser plug-in.

onConnectionStateChange(connectionState, host, datacenter, vmId, userRequested, reason)

The `onConnectionStateChange()` event is generated in response to a change in the connection state.

Parameter Name	Type	Description/Notes
connectionState	string	New connection state. Can be either VMRC_CS_CONNECTED or VMRC_CS_DISCONNECTED.
host	string	Remote host name for the connection.
datacenter	string	Datacenter associated with the remote host.
vmId	string	Virtual machine ID for the current connection.
userRequested	boolean	Boolean value denoting whether the user requested the change in connection state.
reason	string	String containing information on the connection state change. The reason string contains user-visible information, suitable for a dialog or other alert.

onMessage(type, message)

The onMessage() event is generated in response to messages from the VMRC browser plug-in. The message string included with the onMessage() event contains information and other messages from the VMRC browser plug-in that pertain directly to the user of your web application. Your web application can display the message string to the user using a dialog or other alert.

Parameter Name	Type	Description/Notes
type	Property Value Constant	Message type. Values can include VMRC_MESSAGE_INFO, VMRC_MESSAGE_WARNING, VMRC_MESSAGE_ERROR, or VMRC_MESSAGE_HINT
message	string	The message content.

MKS Mode Events

The VMRC browser plug-in generates MKS mode events in response to changes in the console screen or input state. MKS events are generated only if the VMRC browser plug-in is started in MKS mode.

onScreenSizeChange(width, height)

The onScreenSizeChange() event is generated in response to changes in the screen size of the currently connected virtual machine.

Parameter Name	Type	Description/Notes
width	int	New screen width in pixels.
height	int	New screen height in pixels.

onFullscreenChange(fullscreenState)

The onFullscreenChange() event is generated when the VMRC browser plug-in exits or enters full-screen mode.

Parameter Name	Type	Description/Notes
fullscreenState	boolean	If true, the plug-in has entered full-screen mode. If false, the plug-in has exited full-screen mode.

onGrabStateChange(grabState)

The `onGrabStateChange()` event is generated when the VMRC browser plug-in input grab state, when guest input is either grabbed or released.

Parameter Name	Type	Description/Notes
`grabState`	Property Value Constant	New grab state. Values can include `VMRC_GS_GRABBED`, `VMRC_GS_UNGRABBED_HARD`, or `VMRC_GS_UNGRABBED_SOFT`. In a soft-ungrab state, input is redirected to the guest when the user mouses over the guest window.

Devices Mode Events

The VMRC browser plug-in generates Devices mode events in response to changes in the state of currently connected devices, or changes in which devices are present either on the physical client or remote virtual machines. Devices mode events are only generated when the VMRC browser plug-in is started in Devices mode.

onDeviceStateChange(deviceState, host, datacenter, vm, virtualKey, physicalKey, userRequested, reason)

The `onDeviceStateChange()` event is generated in response to a change in the connection state of a virtual machine device.

Parameter Name	Type	Description/Notes
`ds`	string	New device state. Can be either `VMRC_DS_CONNECTED` or `VMRC_DS_DISCONNECTED`.
`host`	string	Host name of the remote host.
`datacenter`	string	Datacenter associated with the remote host.
`vmId`	string	ID of the virtual machine on which the device state changed.
`virtualKey`	string	Virtual device key for which the state changed.
`physicalKey`	string	Physical device key, if the device is connected.
`userRequested`	boolean	Boolean value denoting whether the user requested the change in device state.
`reason`	string	String containing information on the device state change. The `reason` string contains user-visible information, suitable for a dialog or other alert.

onVirtualDevicesChange()

The `onVirtualDevicesChange()` event is generated when there is a change to any virtual devices that are present on the target virtual machine, for any reason. This event contains no parameters.

When your Web application receives the `onVirtualDevicesChange()` event, you can use the `getVirtualDevices()` API method to obtain the updated list of virtual device keys, and the `getVirtualDeviceDetails()` API method to obtain updated detailed information on each virtual device. See "Calling VMRC API Methods on a Virtual Machine" on page 17.

onPhysicalClientDevicesChange()

The `onPhysicalClientDevicesChange()` event is generated when there is a change to any physical devices that are present on the local client machine, for any reason. This event contains no parameters.

When your Web application receives the `onPhysicalClientDevicesChange()` event, you can use the `getPhysicalClientDevices()` API method to obtain the updated list of physical device keys, and the `getPhysicalClientDeviceDetails()` API method to obtain updated detailed information on each physical client device. See "Calling VMRC API Methods on a Virtual Machine" on page 17.

Index

R

S

U

V

W