HTML Console SDK Programming Guide

VMware HTML Console 2.2

vmware®

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

https://docs.vmware.com/

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

Contents

About This Book

The *VMware HTML Console SDK Programming Guide* explains how to use the HTML Console SDK API to write browser-based applications for VMware vSphere® and other VMware products.

Intended Audience

This book is intended for anyone who needs to develop applications containing a remote console. 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.

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 1-1. Revision History

Revision Date	Description
3 July 2023	Changed some chapter and section titles for search engine optimization.
9 November 2022	Revised for the 2.2 release with code sample for remote copy.
17 April 2017	Revised manual for incorporation in the vSphere documentation center.
27 October 2016	No changes for the 2.1 release.
12 April 2016	Very lightly revised for the 2.0 release.
15 September 2015	Initial release of the *VMware HTML Console Programming Guide*.

Related Documentation

Documentation for the HTML Console SDK is available on the VMware developer documentation website. Go to http://developer.vmware.com/sdks/.

Background information for the tasks discussed in this book is available in the vSphere documentation set, for both ESXi hosts and vCenter Server. Go to https://docs.vmware.com.

The VMware HTML Console

1

You can use the HTML Console SDK to connect a Web browser with the user interface of a virtual machine.

This chapter includes the following topics:

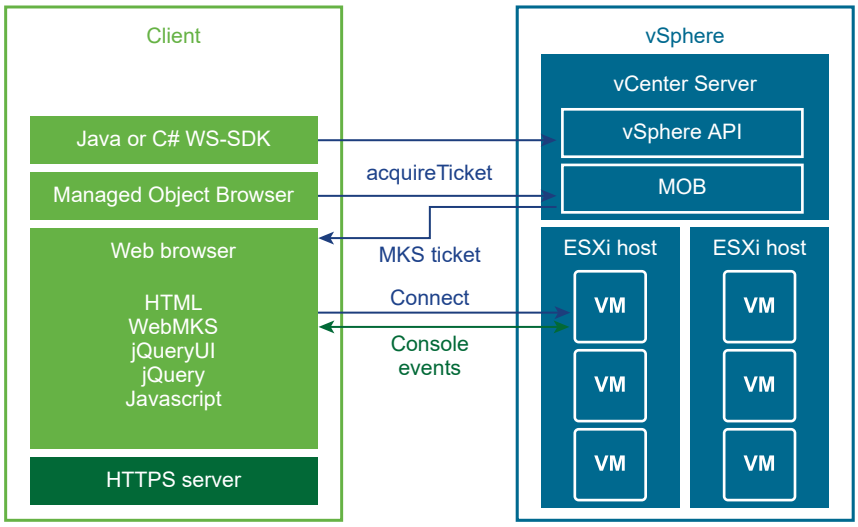
- Architecture of the HTML Console
- Compared to Other VMware Consoles

Architecture of the HTML Console

The HTML Console is a remote console implemented in JavaScript to manage mouse, keyboard, and screen (MKS) events. HTML Console replaces most features of VMRC, the VMware remote console application.

When used with vSphere, your client Web browser connects to a virtual machine console, as shown in WebMKS Architecture.

Figure 1-1. WebMKS Architecture

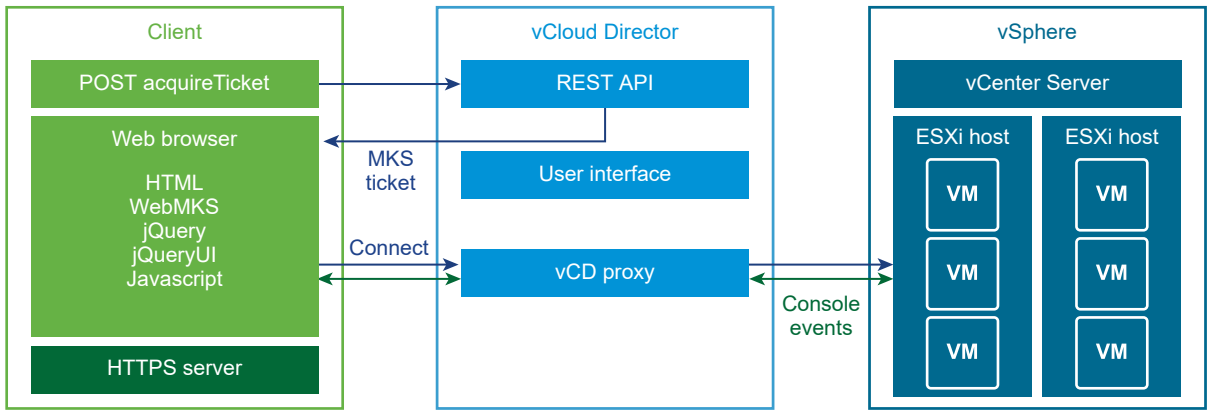


To display a remote console, the client system uses a Web browser to open an HTML file. This file loads the WebMKS Javascript application after calling the jQueryUI and jQuery libraries. Connecting to a virtual machine on vSphere requires an MKS ticket. You can obtain a ticket from the vCenter Server that manages the ESXi host where the virtual machine resides.

After the ESXi host authenticates the MKS ticket, a remote console appears and the Javascript application issues console commands by calling various methods. WebMKS is a VMware implementation to handle remote console events, written in JavaScript for HTML5 compliant browsers. WebMKS is also called WMKS.

When used with vCloud Director, the client communicates with the target virtual machine through the vCloud Director console proxy, as shown in WebMKS with vCloud Director. See the *vCloud API Programming Guide* for an example that shows how to display the virtual machine console.

Figure 1-2. WebMKS with vCloud Director



Remote HTML Console Features

The HTML Console SDK allows developers to add a virtual machine console to an existing Web interface running on VMware vSphere or vCloud Director. With this SDK, end users can interact with the virtual machine console by typing on the keyboard and moving the mouse.

The remote access console can display either a command line or a graphical user interface. It accepts keyboard and mouse or touch screen input, including function keys and on-screen buttons. Keyboard layout selections include French, German, Italian, Japanese, Portuguese, Spanish, Swiss-French, Swiss-German, and possibly other locals. The console display is updated according to mouse movement, typing input, and gestures on your client machine. Gestures include touchscreen scrolling and button tap on a mobile device.

The HTML Console SDK API contains various methods that connect to and communicate with a virtual machine. It also triggers events to notify users of changes to the virtual machine's state. You program these methods and callbacks so users can remotely manage a virtual machine from any system with the appropriate Web browser. Methods include console commands for changing screen mode, sending special key sequences to the virtual machine, and connecting client devices to the virtual machine. The API implements callbacks that communicate back and forth when the user creates console events, or when the virtual machine responds to user input.

The HTML Console SDK is written in JavaScript, based on jQuery and jQueryUI. The jQuery library simplifies navigation and event handling, while jQueryUI provides visual effects such as those in a window system. These libraries can be called from the public Web, or bundled with your application to run in secure environments. WebMKS provides mouse, keyboard, and screen access to a remote console from any supported Web browser. In the SDK, the minimized Javascript file `wmks.min.js` is the WebMKS implementation.

Compared to Other VMware Consoles

The HTML Console is not the only VMware virtual machine console.

The HTML Console SDK is a tool that customers and developers can use to implement remote virtual machine access in a Web application. It differs from earlier remote consoles, and ones built into vSphere.

Compared to the VMRC SDK Plug-in

With the release of vSphere 5.1 in 2012, the VMware Remote Console (VMRC) provided a way to insert a virtual machine console into a Web application. On Windows, the VMRC SDK contained an executable Win32 plug-in for Internet Explorer. On Linux, the SDK contained a 32-bit or 64-bit NPAPI shared object for Mozilla Firefox and Google Chrome. NPAPI was the Netscape plug-in API, now discontinued.

Due to the discontinuation of NPAPI and decreased use of Internet Explorer, alternatives were needed. The final VMRC release was for vSphere 6.0 in 2015.

Compared to vSphere Web Client Consoles

In the vSphere Web Client view of a virtual machine, Summary tab, click the small image of a console to open a browser tab showing a full image of the console screen. If the console is larger than your browser, scroll bars appear. This remote console implementation is similar to WebMKS, but not identical.

In the vSphere Web Client view of a virtual machine, click **Launch Remote Console** to open a stand-alone application for Windows, Mac OS X, or Linux. VMRC appears on upper left in the title bar. This application is not the discontinued VMRC SDK, but is descended from the remote console on VMware Workstation. The stand-alone VMRC offers features not available in the browser console, such as power operations and device management.

Getting Started with the HTML Console SDK

2

After downloading the SDK, you can see how it works with a simple example. You add Javascript code to an HTML file and start a remote console in a Web browser.

This chapter includes the following topics:

Download and Install

The SDK is available in the Downloads section of the VMware {Code} landing page for the HTML Console SDK. You can use your My VMware account or Partner login to authenticate for download.

The SDK is delivered as a ZIP file. The unzipped SDK includes the WebMKS Javascript program with CSS and image folders.

```
SDK:
WebMKS_SDK_2.x.0.zip  css  img  wmks.min.js
```

```
SDK/css:
extended-keypad.css  main-ui.css  trackpad.css  wmks-all.css
```

```
SDK/img:
touch_sprite.png  touch_sprite_feedback.png
```

The `wmks.min.js` program has been minimized so it is not human readable. You can find various Web sites to make the Javascript readable, such as jsbeautifier.com, unminify.com, and the Javascript beautifier.

The WebMKS program requires jQuery, a popular Javascript library, and jQueryUI, a related library for user interfaces. Several versions of these libraries are available on public Web sites, such as jquery.com and ajax.googleapis.com. For secure environments you can download these libraries to run in a datacenter without Internet access.

Because the WebMKS program is written in Javascript, it requires a modern browser with Javascript support. Recent versions of Google Chrome, Mozilla Firefox, Apple Safari, and Microsoft Edge all qualify. If Javascript is enabled, the jQuery and jQueryUI libraries can run.

To see how it works, place the WebMKS program on an HTML page as a script, after loading the jQuery and jQueryUI libraries.

Steps to Start a Console

To create an HTML console, load the WebMKS program as a script into an HTML page, after loading the jQuery and jQueryUI libraries.

Prerequisites

- Your machine must have connectivity to vCenter Server and an ESXi host at a known IP address or FQDN.
- Your machine should have a local HTTP server. Two alternatives are Lighttpd and XAMPP.
- You need locate and possibly download the jQuery and jQueryUI libraries.

Procedure

1. Download the HTML Console SDK from code.vmware.com/sdk/html-console.

   The WebMKS_*.zip file contains the SDK components.
2. Unzip the archive.

   The `wmks.min.js` program appears, along with `css` and `img` folders.
3. Create a console display page using HTML from the next section.
4. If lacking security certificates, permit unsafe connections to vCenter Server and ESXi host.
5. Get a `webmks` ticket using one of the different methods at the end of this chapter.
6. On the `wmks.connect` line, fill in the IP address of the ESXi host, and the VM's `webmks` ticket.

Results

When you open the HTML file in a browser, the VM console should appear.

Sample Javascript Code

A short Javascript application can create a remote HTML Console.

The sample HTML file starts with normal items including a window title. At the beginning of the body it calls `css/wmks-all.css`, which is a compendium of other CSS format files for the keyboard, trackpad, and screen UI. It then calls Web versions of the jQuery and jQueryUI libraries, and the `wmks.min.js` program delivered in the SDK.

The HTML console is placed on a Web page by the `wmksContainer` object, and registered for keyboard and mouse events. After setup, the connect call is made to a Web service secure (`wss`) URL. You will have to supply this URL manually; it is not automated in the sample code.

Example: HTML file to display remote console.

```
<!DOCTYPE html PUBLIC"-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>Console</title>
</head>
<body>
<link rel="stylesheet" type="text/css" href="css/wmks-all.css" />
<script type="text/javascript" src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
<script type="text/javascript" src="http://code.jquery.com/ui/1.12.0/jquery-ui.min.js"></
script>
<script type="text/javascript" src="wmks.min.js"></script>
<div id="wmksContainer" style="position:absolute;width:100%;height:100%"></div>
<script>
var wmks = WMKS.createWMKS("wmksContainer",{})
  .register(WMKS.CONST.Events.CONNECTION_STATE_CHANGE, function(event,data){
    if(data.state == WMKS.CONST.ConnectionState.CONNECTED){
      console.log("connection state change : connected");}
  });
wmks.connect("wss://ESXi.host.IP.Address:443/ticket/webmksTicket");
</script>
</body>
</html>
```

This HTML file should be on a machine with Web service, for instance XAMPP or Lighttpd.

Replace *ESXi.host.IP.Address* with the IP address or fully-qualified hostname of the ESXi host where the virtual machine resides. Your browser must have accepted the security certificate from this ESXi host.

Acquire MKS Ticket with a Browser

To get the *webmksTicket*, find the vCenter Server that manages the ESXi host. You can get a ticket using the vSphere API (see Acquire MKS Ticket with vSphere API) or with the managed object browser (MOB).

To get a ticket with the MOB:

1 In a browser go to `https://vCenter.Server.IP.Address/mob`

2 Provide your login credentials for Administrator@vsphere.local or another SSO account

3 Click **content** > **rootFolder** (group-dl) > **Datacenter** > **vmFolder**

4 Click the VM number of the virtual machine you want to connect to

5 Under Methods, click **AcquireTicket**

6 Type **`webmks`** in the value field and click **`Invoke Method`**

7 Click to show. Copy and paste the ticket string and replace *webmksTicket*

You can also get a *webmksTicket* from the ESXi host, but the MOB must be enabled on the ESXi host, as described in KB 2108405, https://kb.vmware.com/kb/2108405.

Once you know the managed object ID (`moid`) of the virtual machine, you can quickly get another ticket at the following URL. Replace *NN* with the VM number:

```
https://vCenter.Server.IP.Address/mob/?moid=vm-NN&method=acquireTicket
```

If you have Windows PowerShell with PowerCLI installed, you can get a ticket as follows:

```
PowerCLI C:\> $vm.ExtensionData.AcquireTicket("webmks")
```

Acquire MKS Ticket with vSphere API

You can get the *webmksTicket* using a programming language that the Web Services SDK supports, such as Java or C#.

The `acquireTicket` method creates and returns a one-time credential used to establish a connection to a virtual machine. The `webmks` ticket type verifies a remote mouse-keyboard-screen connection. It requires **VirtualMachine.Interact.ConsoleInteract** privilege.

The following Java code returns an MKS ticket, given the managed object reference of a virtual machine and ticket type `webmks`.

```
public
VirtualMachineTicket acquireTicket(final ManagedObjectReference vmMoRef, String ticketType)
throws Exception {
   VirtualMachineTicket vmTicket = null;
   vmTicket = super.getPortType().acquireTicket(vmMor, ticketType);
   return vmTicket;
}
```

To acquire the MKS ticket with vCloud Director, you can use a REST API call such as:

```
POST https://vcloud.example.com/api/vApp/vm-NN/screen/action/acquireMksTicket
```

Run the HTML Console Sample Code

The sample HTML code should reside on a machine with an HTTP server, in a directory prescribed for Web service. Communicating machines must have accepted security certificates from each other. To use the HTML console, navigate to the location of the sample code.

Once the remote console starts, log in to the virtual machine on its ESXi host, if a login session was not already started.

When you are finished with the HTML Console, you can disconnect and destroy the WMKS component. Both happen automatically when the HTML sample code page is closed.

The Create WebMKS Method and Options

3

Create WebMKS or `createWMKS` is the first method to call when using the HTML Console SDK. This method generates the console widget, which displays the remote screen, and returns the WMKS core object. The core object which is later associated with a in the document object model (DOM). After connecting to a virtual machine, your application can use the available APIs to perform operations with the remote console.

This chapter includes the following topics:

createMKS Parameters

The `createWMKS` method takes two parameters and returns the WMKS core object, used in subsequent operations. The first parameter is the name of a element. The second parameter is a possibly empty list of options.

id

Type: string. The ID of a element that is the container of the canvas used to show remote screens.

options

Type: JSON object. Options affect WMKS behavior, as documented here. If not specified, option defaults are used.

In the following example, this method creates a WMKS core object named "container" with all default options, and returns the WMKS core object in variable `wmks`.

```
var wmks = WMKS.createWMKS("container",{});
```

createMKS Options

Multiple options control the behavior of WMKS.

Option	Description	Type	Notes
audioEncodeType	Indicates the type of audio encoding method being used. Possible values for WMKS.CONST.AudioEncodeType are: `vorbis`, `opus`, or `aac`.	enum	
changeResolution	If true, WMKS sends a change resolution request to the connected VM. The requested resolution matches the container size. If the request fails, use the rescale and position operations to change the resolution manually.	Boolean	Default is true.
enableUint8Utf8	If true, enables the `uint8utf8` protocol for projects that do not support the binary protocol.	Boolean	Default is false.
fixANSIEquivalentKeys	If true, enables translation of non-ANSI US keyboard layouts to ANSI US keyboard layout equivalents. Tries to match keys on the international keyboard to keys in different locations or with different Shift status on the US keyboard.	Boolean	Default is false.
ignoredRawKeyCodes	Ignores the keycodes. Do not send keycodes to the server.	array	Default is empty.
keyboardLayoutId	Provides different language keyboard setups for the Guest OS. Remote desktop and local desktop keyboard layouts must be the same. For international mapping, WMKS supports vScancode. This option does not support mobile devices. See keyBoardLayoutId Details	string	Default value is en-US.
position	Indicates where the remote screen should appear in the container. Values are either center or top left.	enum	Default value is WMKS.CONST.Position.CENTER
rescale	Indicates whether to rescale the remote screen to fit the container size.	Boolean	Default is true.
retryConnectionInterval	The interval in milliseconds before attempting to reconnect the Web client and server after a failed attempt. If less than 0, WMKS does not attempt to create the connection again.	integer	Default value is -1.
reverseScrollY	If true, sends the opposite value of the mouse when to the connected VM. This is for touch devices that scroll in the opposite direction.	Boolean	Default is false.
sendProperMouseWheelDeltas	If true, actual mouse wheel event delta values are sent from the browser to the server. If unspecified, normalized event deltas values are either: -1, 0, or 1.	Boolean	Default is false.

Option	Description	Type	Notes
useNativePixels	Enables the use of native pixel sizes on the device. For example, on the iPhone 4+ or iPad 3+ devices, the true setting enables Retina mode, providing more screen space for the guest and making everything appear smaller.	Boolean	Default is false.
useUnicodeKeyboardinput	If true, WMKS attempts to send Unicode messages from the user to the server. If unspecified, WMKS sends messages using either Unicode or keyboard scan codes.	Boolean	Default is false.
useVNCHandshake	Enables a standard VNC handshake. Implement this option when the endpoint uses standard VNC authentication. Set to false if connecting to a proxy that uses `authd` for authentication and does not perform a VNC handshake.	Boolean	Default is true.
VCDProxyHandshakeVmxPath	The string passed by the VNC protocol. If a connection request for a VMX path is received, the VNC protocol responds with the VCDProxyHandshakeVmxPath when connecting to vCloud Director.	string	Default value is null.

keyBoardLayoutId Details

The keyBoardLayoutId option supports the following browsers:

- For Windows: Internet Explorer, Firefox, and Chrome.
- For Mac OS X: Chrome and Safari.

To support different languages, both the local machine with browser and the remote virtual machine must have and be set to the correct locale. For example, when a user chooses German in the keyboard select list, both the local machine and the remote VM must be running a German input method editor (IME).

So users can select the language they prefer, you provide a selection list in HTML.

```
<select id="selectLanguage">
   <option value="en-US">English</option>
   <option value="ja-JP_106/109">Japanese</option>
   <option value="de-DE">German</option>
   <option value="it-IT">Italian</option>
   <option value="es-ES">Spanish</option>
   <option value="pt-PT">Portuguese</option>
   <option value="fr-FR">French</option>
   <option value="fr-CH">Swiss-French</option>
   <option value="de-CH">Swiss-German</option>
</select>
```

Then call the `setOption` API in a Javascript file as in the following example.

```
$('#selectLanguage').change(function(){
    if(!wmks) return;
    var keyboardLayoutId = $(this).find(":selected").val();
    wmks.setOption('keyboardLayoutId',keyboardLayoutId);
 });
```

Handling Events with the HTML Console

4

The HTML Console generates events when the state of the currently connected VM changes, or in response to messages from the currently connected VM.

This chapter includes the following topics:

Event Binding in jQuery

As stated in the introduction, the HTML Console SDK is based on jQuery. HTML Console applications can use the jQuery `.on()` method to associate screen events with Javascript actions.

Since version 1.7, jQuery provides the .on() method for responding to an event on the selected elements. This is called an event binding. The jQuery model works with events causing changes to elements in the HTML document object model (DOM).

```
          Event Handler ------------------> Change element in the DOM
$(object).on('event', function() {
   ...                                      $("button".attr(...)
});
```

For example, an HTML Console application can implement screen resize as follows:

```
$(window).on('resize', function(e) {
   updateSize();
   updateScreen();
});
```

Setting the Event Handler

The HTML Console SDK associates a jQuery `.on()` event with a WMKS event handler.

The WMKS object uses the HTML Console SDK `connect()` method to set up a WebSocket URL and grab the remote console of a VM. To receive event notifications from the VM, you then call the HTML Console SDK `register()` method to set up the event handler. See Event Handler API Methods.

The following table shows the events defined by `WMKS.CONST.Events`.

Event Handlers

Each event handler processes one type of event with specific data.

event

jQuery event information received

data

data collected about the event

Event	Description	Data Collected
audio	Generated when an audio message is received from the server.	Values: ■ sampleRate ■ numChannels ■ containerSize ■ sampleSize ■ length ■ audioTimestampLo ■ audioTimestampHi ■ frameTimestampLo ■ frameTimestampHi ■ flags ■ data
connectionstatechange	Generated in response to a change in the connection state of the VM.	ConnectionState values: ■ connecting ■ connected ■ disconnecting If state is WMKS.CONST.ConnectionState.CONNECTING, data collected includes: `vvc` and `vvcSession`. If state is WMKS.CONST.ConnectionState.DISCONNECTED, data collected includes: `reason` and `code`.
copy	Generated when the server sends a cut text (copy) event.	Type: string. Value of the copied string.
error	Generated when an error occurs.	ErrorType values: ■ authenticationfailed ■ websocketerror ■ protocolerror

Event	Description	Data Collected
fullscreenchange	Generated when the WMKS console enters or exits full screen mode.	Type: Boolean with values: ■ True: entering full screen. ■ False: exiting full screen.
heartbeat	Once a connection is established, the server generates heartbeat events.	Type: integer with value of the heartbeat interval in seconds. The client listens for the heartbeat. If two intervals pass without a heartbeat, the connection is dead.
keyboardledschanged	Generated when the keyboard LED lock state changes on the remote VM.	Type: integer with values: ■ 1: scroll lock ■ 2: num lock ■ 4: caps lock
screensizechange	Generated in response to changes in the screen size of the connected VM.	Type: integer with values: ■ pixel width of the connected VM console ■ pixel height of the connected VM console
toggle	Generated when an input device is shown or hidden.	InputDeviceType values: ■ KEYBOARD ■ EXTENDED_KEYPAD ■ TRACKPAD Boolean type values: ■ True: shown ■ False: hidden

Copy from Remote Desktop

This section presents sample Javascript code to implement copy-paste from a remote virtual machine to the local client console.

Prerequisite: Enable clipboard copy from the remote to local client.

After copy from the remote desktop, clipboard data becomes available after the mouse cursor is no longer focused on the remote desktop.

To implement remote copy, add this Javascript code as a script to existing HTML console code:

```
<script>
   $(function() {
      var container = $("#container");
      var wmks = WMKS.createWMKS("container", {});
      wmks.register(WMKS.CONST.Events.CONNECTION_STATE_CHANGE, function(evt, data) {
         switch (data.state) {
            case WMKS.CONST.ConnectionState.CONNECTING:
               console.log("The console is connecting");
               break;
            case WMKS.CONST.ConnectionState.CONNECTED:
               console.log("The console has been connected");
               // need to send grab first, bind on the canvas element
               $("#mainCanvas").focus(function() {
```

```
                  wmks.grab();
               });
               $("#mainCanvas").blur(function() {
                  wmks.ungrab();
               });
               break;
            case WMKS.CONST.ConnectionState.DISCONNECTED:
               console.log("The console has been disconnected");
               break;
         }
      });
      wmks.register(WMKS.CONST.Events.ERROR, function(evt, data) {
         console.log("Error: " + data.errorType);
      });
      wmks.register(WMKS.CONST.Events.COPY, function(evt, data) {
         // here is the remote clipboard data:
         console.log(evt, data);
      });
      wmks.connect("wss://testurl");
   });
</script>
```

Make sure the remote desktop has already enabled clipboard feature. Before ungrab, a grab event is necessary. You can get remote clipboard data from the server after sending ungrab.

HTML Console SDK Methods

5

The WMKS core object uses HTML Console SDK methods to connect to and perform operations on a remote VM console. HTML Console SDK methods below are grouped by related function.

Each method definition includes required parameters, return values, and an example call. In example calls, the WMKS core object is named `wmks`. WMKS Constants provides descriptions and values of parameter constants.

This chapter includes the following topics:

General API Methods

General purpose API methods provide information about WMKS. These methods can be called at any time before or during connection to a target VM.

getVersion()

Retrieves the current version number of the HTML Console SDK.

Parameters

None

Return Value

Type: string. Contains the full version number of the HTML Console SDK.

Example Call

```
var version = wmks.getVersion();
```

getConnectionState()

Retrieves the current connection state.

Parameters

None

Return Value

Type: string. Any constant value for WMKS.CONST.ConnectionState. See WMKS Constants.

Example Call

```
var version = wmks.getConnectionState();
```

Lifecycle API Methods

The WMKS core object uses lifecycle API methods to manage the connection with the target VM.

connect()

Connects the WMKS to a remote virtual machine using the WebSocket URL and sets up the UI.

Parameters

Type: string. WebSocket URL in format:

```
<ws|wss>://<host:port>/<path>/?<authentication info>
```

Return Value

None

Example Call

```
wmks.connect("ws://localhost:8080");
```

disconnect()

Disconnects from the remote virtual machine and exits the UI.

Parameters

None

Return Value

None

Example Call

```
wmks.disconnect();
```

destroy()

Terminates any active connection with the VM and removes the widget from the associated element. Always call this method before removing a WMKS element from the HTML document object model (DOM).

Parameters

None

Return Value

None

Example Call

```
wmks.destroy();
```

Display API Methods

The WMKS core object uses display API methods to control the remote screen size.

setRemoteScreenSize()

Sends a request to set the screen resolution of the currently connected VM console. If the requested width and height parameters are larger than the allocated size of the WMKS widget, the sizing is normalized.

Parameter1

Type: integer. Width of the screen in pixels.

Parameter2

Type: integer. Height of the screen in pixels.

Return Value

None

Example Call

```
wmks.setRemoteScreenSize(800,600);
```

getRemoteScreenSize()

Retrieves the screen width and height in pixels.

Parameter

None

Return Value

Object with format {width:*w*, height:*h* }

Example Call

```
var size = wmks.getRemoteScreenSize;
```

updateScreen()

Changes the resolution or rescales the remote screen to match the container size. Behavior depends on settings for changeResolution, rescale, and position options described in createMKS Options.

- If changeResolution is true, the remote screen resolution changes to match the container size.
- If changeResolution is false, check the rescale option. If true, the remote screen rescales to match the container size.
- If both changeResolution and rescale are false, check the position option. Depending on the value, the remote screen appears in the container at either the center or top left.

Parameter

None

Return Value

None

Example Call

```
wmks.updateScreen();
```

Full Screen API Methods

The WMKS core object uses full screen API methods to control full screen mode in the browser.

canFullScreen()

Returns whether the browser is enabled for full screen mode. Due to security concerns with keyboard input in full screen mode, Safari is not enabled for full screen mode.

Parameters

None

Return Value

Type: Boolean. True indicates that the browser is enabled for full screen mode.

Example Call

```
wmks.canFullScreen();
```

isFullScreen()

Returns whether the browser is in full screen mode.

Parameters

None

Return Value

Type: Boolean. True indicates that the browser is in full screen mode.

Example Call

```
wmks.isFullScreen();
```

enterFullScreen()

Forces the browser to enter full screen mode. In full screen mode, only the remote screen appears.

Parameters

None

Return Value

None

Example Call

```
wmks.enterFullScreen();
```

exitFullScreen()

Forces the browser to exit full screen mode.

Parameters

None

Return Value

None

Example Call

```
wmks.exitFullScreen();
```

Input API Methods

The WMKS core object uses input API methods to send keyboard input to the server or VM.

sendInputString()

Sends a string as keyboard input to the server.

Parameters

Type: string.

Return Value

None

Example Call

```
wmks.sendingInputString("test");
```

sendKeyCodes()

Sends a series of special key codes to the VM. The input is an array of key codes and the method sends keydown events for each key code in the order listed. This method sends keyup events for each in reverse order. Use this method to send key combinations such as Alt+Tab.

For a list of key codes, see Javascript Key Codes.

Parameters

Type: array. Each element in the array can be a keycode or a Unicode character, where keycode is for non-printable keys, and Unicode is a negative number for printable characters. With Unicode, a negative integer such as -118 represents the Latin small letter v.

Return Value

None

Example Call

```
wmks.sendKeyCodes([18,9]) ; // Alt + Tab
```

sendCAD()

Sends a Control+Alt+Delete key sequence to the connected VM.

Parameters

None

Return Value

None

Example Call

```
wmks.sendCAD();
```

Mobile API Methods

The WMKS core object uses mobile input methods to control the input and display on a mobile device. The input device types are keyboard, extended keyboard, and trackpad.

enableInputDevice()

Enables and initializes the input device type on a mobile device.

Parameters

Any constant value for WMKS.CONST.InputDeviceType. See WMKS Constants.

Return Value

None

Example Call

```
wmks.enableInputDevice(WMKS.CONST.InputDeviceType.KEYBOARD);
```

disableInputDevice()

Disables and terminates the input device type on a mobile device.

Parameters

Any constant value for WMKS.CONST.InputDeviceType. See WMKS Constants.

Return Value

None

Example Call

```
wmks.disableInputDevice(WMKS.CONST.InputDeviceType.KEYBOARD);
```

showKeyboard()

Shows the keyboard on a mobile device.

Parameters

None

Return Value

None

Example Call

```
wmks.showKeyboard();
```

hideKeyboard()

Hides the keyboard on a mobile device.

Parameters

None

Return Value

None

Example Call

```
wmks.hideKeyboard();
```

toggleExtendedKeypad()

Depending on availability, shows or hides the extended keyboard on a mobile device such as the iPhone 6s or 6.

Parameters

A map such as minToggleTime(50) or {minToggleTime: 50}. If called frequently with the time between subsequent calls less than the minToggleTime in milliseconds, the call is not executed.

Return Value

None

Example Call

```
wmks.toggleExtendedKeypad({minToggleTime: 50});
```

toggleTrackpad()

Depending on current visibility, shows or hides the trackpad on a mobile device.

Parameters

A map such as minToggleTime(50) or {minToggleTime: 50}. If called frequently with the time between subsequent calls less than the minToggleTime in milliseconds, the call is not executed.

Return Value

None

Example Call

```
wmks.toggleTrackpad({minToggleTime: 50});
```

toggleRelativePad()

Depending on current visibility, shows or hides the relative trackpad on a mobile device.

WebMKS adds a toggleRelativeMouse button in the UI and binds it to the toggleRelativePad() API. If the button is clicked, the relative trackpad appears and sends relative mouse events to the remote guest. Use this method when the remote guest OS does not install VMware Tools and cannot accept an absolute mouse event.

To display the relative trackpad, the SDK includes a wmks-all.css file provided in the css and img folders.

Parameters

A map such as minToggleTime(50) or {minToggleTime: 50}. If called frequently with the time between subsequent calls less than the minToggleTime in milliseconds, the call is not executed.

Return Value

None

Example Call

```
wmks.toggleRelativePad({minToggleTime: 50});
```

setOption API Method

The WMKS core object uses the `setOption` method to change option values for `createWMKS`.

Of the options listed in createMKS Options, this method supports:

- rescale
- position
- changeResolution
- useNativePixels
- reverseScrollY
- fixANSIEquivalentKeys
- sendProperMouseWheelDeltas
- keyboardLayoutId

Parameter1

Type: string. optionName

Parameter2

Type: Boolean. optionValue

Return Value

None

Example Call

```
wmks.setOption("changeResolution",false);
```

Event Handler API Methods

Event handler API methods add and remove handlers for WMKS events.

See Chapter 4 Handling Events with the HTML Console.

register()

Register the event handler for the WMKS.

Parameter1

Any constant value for WebMKS.CONST.Events. See WMKS Constants.

Parameter2

eventHandler (javascript callback function)

Return Value

None

Example Call

```
var connectionStateHandler = function(event,data){};
wmks.register(WebMKS.CONST.Events.CONNECTION_STATE_CHANGE,
              connectionStateHandler);
```

unregister()

Unregister the event handler for the WMKS.

- If called without parameters, it removes all event handlers.
- If called with one parameter, it removes event handlers for the given event name.
- If no such event was registered, it has no effect.

Parameter1

Any constant value for WebMKS.Events. See WMKS Constants.

Parameter2

eventHandler (javascriptCallbackFunction)

Return Value

None

Example Call

```
wmks.unregister(WebMKS.CONST.Events.CONNECTION_STATE_CHANGE,
                connectionStateHandler);
```

WMKS Constants

HTML Console SDK methods use the following constants for parameter input and return values. Each description includes an example that lists the constant values.

AudioEncodeType

Types of audio encode methods.

```
AudioEncodeType: {
        VORBIS: "vorbis",
        OPUS: "opus",
        AAC: "aac"
},
```

ConnectionState

Describes the state when attempting to connect to a remote VM.

```
ConnectionState: {
        CONNECTING: "connecting",
        CONNECTED: "connected",
        DISCONNECTED: "disconnected"
},
```

ErrorType

Errors that can occur when WMKS is connected to a remote VM.

```
ErrorType: {
        AUTHENTICATION_FAILED: "authenticationfailed",
        WEBSOCKET_ERROR: "websocketerror",
        PROTOCOL_ERROR: "protocolerror"
},
```

Events

Events that WMKS can trigger.

```
Events: {
        CONNECTION_STATE_CHANGE: "connectionstatechange",
        REMOTE_SCREEN_SIZE_CHANGE: "screensizechange",
        FULL_SCREEN_CHANGE: "fullscreenchange",
        ERROR: "error",
        KEYBOARD_LEDS_CHANGE: "keyboardledschanged",
        HEARTBEAT: "heartbeat",
        AUDIO: "audio",
        COPY: "copy",
        TOGGLE: "toggle"
},
```

InputDeviceType

Specifies the input device when viewing VM consoles on mobile devices.

```
InputDeviceType: {
        KEYBOARD: 0,
        EXTENDED_KEYBOARD: 1,
        TRACKPAD: 2
}
```

Position

WMKS displays the remote screen of the VM in equal proportions. After rescale, the remote screen size might not match the container size. Position provides two options to position the screen in the container.

```
Position: {
        CENTER:   0,
        LEFT_TOP: 1
},
```

Javascript Key Codes

Javascript assigns a numeric code to every key on a keyboard, including ones that control programs and interfaces.

Alphanumeric Keys

Key codes for alphanumeric and function keys are consecutively numbered. Alphanumeric and symbol keys also have Unicode values, expressed as a negative number with Unicode decimal code.

- The 0 to 9 keys are codes 48 to 57
- The A to Z keys are codes 65 to 90
- Keys on the number pad are codes 96 to 105

- Function keys F1 to F12 are 112 to 123.

For program clarity it is better to use Unicode negative decimal codes than alphanumeric key codes.

Table of Key Codes

These are keys that control the user interface and operating system. Not all keys appear on all keyboards.

Key	Code	Key	Code
Backspace	8	Print Screen	none
Tab	9	Insert	45
Enter	13	Delete	46
Shift	16	Windows Left	91
Control	17	Windows Right	92
Alt	18	Select	93
Break	19	Multiply	106
Caps Lock	20	Add	107
Escape	27	Separator	109
Space Bar	32	Subtract	109
Page Up	33	Decimal Point	110
Page Down	34	Divide	111
End	35	Num Lock	144
Home	36	Scroll Lock	145
Left Arrow	37		
Up Arrow	38		
Right Arrow	39		
Down Arrow	40		