Viewpoint

Using JavaScript in VET Web Applications

Version 1.0

November 16, 2001

© 2001 Viewpoint Corporation. All Rights Reserved.

Using JavaScript in VET Web Applications

Viewpoint Experience Technology (VET), Viewpoint Stream Tuning Studio, Viewpoint Scene Builder, and Viewpoint Media Player (VMP) are registered trademarks or trademarks of Viewpoint Corporation in the United States and in other countries.

Companies, names, and data used in examples herein are fictitious unless otherwise noted. Information in this document is subject to change without notice.

All other product and company names mentioned herein are the trademarks of their respective owners.

Disclaimer
Except as expressly provided otherwise in an agreement between you and Viewpoint, all information, software, and documentation is provided "as is," without warranty of any kind. Viewpoint makes no warranties, express or implied, including without limitation the implied warranties of merchantability and fitness for a particular purpose regarding such information, software and documentation. Viewpoint does not warrant, guaranty, or make any representations regarding the use or the results of the software in terms of its correctness, accuracy, reliability, timeliness, suitability or otherwise. The entire risk as to the results of performance of the software is assumed by you.

In no event will Viewpoint be liable for any special, indirect, consequential, punitive, or exemplary damages or the loss of anticipated profits arising from the performance of the software or resulting from the loss of use, data or profits, whether in an action for breach of contract or warranty or tort (including negligence) arising out of or in connection with the information, technology, software and documentation.

The web site and publications may contain technical inaccuracies or typographical errors. Viewpoint assumes no responsibility for and disclaims all liability for any such inaccuracy, error, or omission in the web site and documentation and in any other referenced or linked documentation. Viewpoint may make changes to the information, software, web site, documentation, prices, technical specifications, and product offerings in its sole discretion at any time and without notice.

Author: Derek Davies
Editor: Carolyn Gronlund
Contributor: Doralee Moynihan

Viewpoint Corporation
498 Seventh Avenue
Suite 1810
New York, NY 10018

Contents

© 2001 Viewpoint Corporation. All rights reserved.

© 2001 Viewpoint Corporation. All rights reserved.

Chapter 1: Introduction

About This Guide

This guide shows some of the things you can do with JavaScript and Viewpoint Experience Technology (VET) XML to script a VET Web application.

This guide includes these chapters:

- Chapter 1: Introduction
  Read an overview of Viewpoint Experience Technology, rich media, VET Web applications, and the authoring tools you can use to create your rich media Web application.
- Chapter 2: Integrating VET into Your Web Pages
  Learn how to use the MTS3 Interface to auto-install Viewpoint Media Player (VMP) and play VET content from your Web page.
- Chapter 3: Controlling XML Animations From JavaScript
  Learn some simple ways to embed JavaScript animation triggers and controls in an .html file, so you can incorporate scene navigation icons and buttons directly into the Web page design.
- Chapter 4: Refresh, Reload, and Resize a Scene From JavaScript
  See how to set the VET scene to refresh/reload from the HTML page, and how to dynamically resize the scene as the Web browser resizes.
- Chapter 5: Setting and Constraining Scene and Object Parameters With JavaScript
  Learn how the VET scene and objects in it can be influenced by parameters chosen from the HTML page.
- Chapter 6: Using JavaScript With Scene Interactors
  Expand your scene interactors to include events from the HTML page.
- Chapter 7: Making a JavaScript Call in a VET Scene
  Use JavaScript calls in a VET scene to display information about the scene on the HTML page.
- Chapter 8: Using Automatically Generated Data in a VET Scene
  Bring data into a scene from many different sources using JavaScript.
- Chapter 9: Help, Resources, and Feedback
  Find out how where can learn more about Viewpoint Experience Technology, get free application and documentation downloads, and get technical support.

Related Documents

You'll find related information on the Viewpoint Developer Central Web site at http://developer.viewpoint.com/.

- If you are new to XML, read *Viewpoint Experience Technology XML Authoring Overview*.
- To learn about XML tags and properties, read *Viewpoint Experience Technology XML Reference Guide*.
- To learn about scripting interaction for VET Web applications, read *Viewpoint Scene Interactors Reference Guide*.

You may also find these guides useful:

- *JavaScript: The Definitive Guide*, by David Flanigan. Published by O'Reilly and Associates, Inc., 1998.
- *HTML and XHTML: The Definitive Guide*, by Chuck Musciano and Bill Kennedy. Published by O'Reilly and Associates, Inc., 2000.

Links to Live Examples and Example Files

At the beginning of each example is a **content link** and **scene files link**. With your computer connected to the Internet,

© 2001 Viewpoint Corporation. All rights reserved.

- Click the content link to load a live example into your default Web browser.
- Click the scene files link to download the actual files for the VET Web application described.

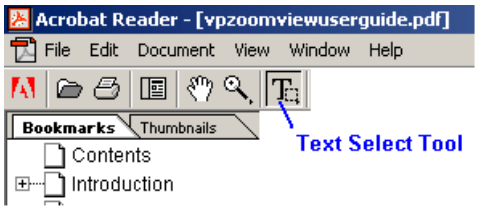
© 2001 Viewpoint Corporation. All rights reserved.

Using Code Samples From This Guide

This guide includes several XML and JavaScript code examples. To quickly implement functionality described in this user guide, you can copy code samples directly from this guide and paste them into your .mtx or .html file.

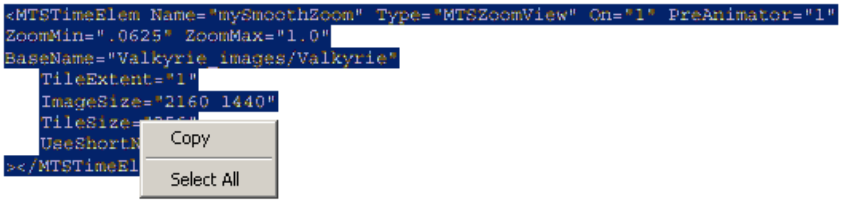
To copy a code sample from the .pdf version of this guide

1 In Adobe® Acrobat® Reader, click the **Text Select Tool** button.



2 Click and drag to highlight the code sample you want.

3 Right-click the highlighted text, and click **Copy**.



What is a VET Web Application?

Until recently, **Viewpoint Experience Technology (VET)** has been described as 3D scenes, rich media content, or VET-enabled Web pages. However, developing for the Web with VET offers much more than those terms imply. With VET you are creating interactive **VET Web applications** that can include XML scripting, JavaScript, and rich media such as 3D models, 2D graphics, animations, sound, and Macromedia® Flash™ files, as well as Viewpoint's proprietary ZoomView and HyperView technologies. VET Web applications are streamed over the Internet and powered by **Viewpoint Media Player (VMP)**, a Web browser plug-in.

A VET Web application can be confined to the VET layer (embedded in a Web page) or can include the entire Web page with JavaScript interactions between the HTML and the VET scene. Web developers using VET can use a full range of content-delivery options, including interactivity and navigation, as well a full range of rich media, including 3D models, 2D graphics, animation, sound, and so on. When planning a VET Web application, decide what objective you want it to accomplish, what features can best meet that objective, and what experience you want your customers to have when interacting with it.

VET Web applications are created using standard authoring tools for 3D modeling, 2D graphics, and so on, as well as Viewpoint tools such as **Viewpoint Scene Builder** (a utility to assemble and edit the content of a Viewpoint scene).

Viewpoint Corporation offers other tools as well to help you bring 3D and rich media content to the Web. To find out more about Viewpoint's family of applications and utilities, go to http://www.viewpoint.com.

Note: VET is free of charge for non-commercial use, allowing you to try before you buy.

© 2001 Viewpoint Corporation. All rights reserved.

About Viewpoint Media Files: .mts, .mtx, .mtz, and .mzv

Viewpoint Media Files—.mts and .mtx formats—can be exported directly from most 3D modeling applications or published from the **Viewpoint Scene Builder**, a utility designed to assemble and edit the content of a Viewpoint scene.

Viewpoint Media Player reads Viewpoint Media Files in order to display a rich media scene:

- .mts file Contains a compressed collection of resources, called "media atoms" (3D objects, material properties, object movies, and environmental lightmaps, to name a few) orchestrated by the .mtx (XML-based) file to create a scene.
- .mtx file An XML-based file that contains the hierarchical relationships between elements in the scene and is the script for staging them.

  Note: Broadcast Key files also use the .mtx filename extension and are typically named bkey.mtx. If you have an old Broadcast Key file with a .txt filename extension, you should rename the file to have an .mtx extension instead.

- .mtz file The compressed, binary version of an .mtx file. An .mtz file can be converted back to .mtx for editing.

  Note: Although the downloadable examples include .mtx files, the HTML links in this document refer to .mtz files, the preferred format for publishing a VET Web application. Edit the .html files to refer to the .mtx file, if you want to customize these examples.

  Look for this code for the MTS3 Interface constructor in the .html file and edit the file name to match the .mtx file name:

```
<!-- FIRST OBJECT/EMBED -->
<script language="javascript">
     vmp = new MTSPlugin("filename.mtz", 400, 400, "", "popUp",
     "GenieMinimumVersion=50332496");
</script>
```

- .mzv file A file format for compressed image tiles (sections of a high-resolution image) used by the Viewpoint ZoomView images.

System and Software Requirements

Creating Viewpoint Experience Technology (VET) Web content requires the following software and systems.

Required Software Applications

- 3D modeling software that supports VET media file export or Viewpoint Scene Builder
- Viewpoint Media Player
- Netscape Navigator 4.07 or later, or Microsoft Internet Explorer 4.x or later
- Any XML text editor (XML Spy® is Viewpoint's preferred editor for the Windows platform)

© 2001 Viewpoint Corporation. All rights reserved.

Viewpoint Media Player Minimum System Requirements

Users viewing your Viewpoint Experience Technology web pages will have the best results if they meet the following minimum system requirements:

- Windows 95, Windows NT 4.x, or Windows XP
- Pentium 166 or faster (Pentium II recommended)
- 5 MB free disk space
- 32MB RAM (64mb recommended)
- Netscape Navigator 4.07 or later (not 6.0), Microsoft Internet Explorer 4.x or later, or AOL 4.0 or later
- 256 color display (24-bit recommended)
- 28.8Kbps modem

© 2001 Viewpoint Corporation. All rights reserved.

Chapter 2:
Integrating VET into Your Web Pages

About the MTS3 Interface: Adding VET Content to a Web Page

With a few lines of code, you can enable your Web page to display VET content. The **Viewpoint MTS3 Interface** makes this possible.

This interface is implemented as a JavaScript constructor and a suite of functions in a file called MTS3Interface.js. Including this file into your HTML page offers a standard for creating Web pages enhanced with VET.

The MTS3 Interface offers these features:

- Provides auto-installation for VMP.
- Manages multiple VET scenes on a single page.
- Handles content meant to display on more than one operating system or browser.
- Supports two-way communication between the HTML and VET environments to incorporate dynamic data and facilitate interaction.
- Posts events (equivalent to the PostMessage action inside the XML).
- Offers error code handling functions.

The code you include in your HTML file is passed to the MTS3 Interface constructor and interpreted for the scene. (The MTS3 Interface constructor is described later in the section called "Calling the MTS3 Interface From Your Web Page.")

Ensuring Browser Compatibility with the MTS3 Interface

The MTS3 Interface provides a uniform standard for Viewpoint JavaScript calls. The class constructor ensures that Viewpoint objects are properly instantiated in Microsoft Internet Explorer and Netscape on Windows and MacOS. It is no longer necessary to have customized JavaScript code for specific browsers and operating systems.

The MTS3 Interface automatically embeds the necessary code (depending on the browser and operating system detected) to guarantee that VMP and the browser work together predictably and efficiently. The MTS3 Interface also handles all calls and messages that are sent back by VMP.

The MTS3 Interface supports these configurations:

- Microsoft Internet Explorer 4.x or higher on Windows 95/98/ME/NT/2000/XP
- Microsoft Internet Explorer 5.x on MacOS 8.5 or later (not OS X)
- Netscape Communicator 4.x or later (not 6.0) on MacOS 8.5 or later (not OS X), Windows 95/98/ME/NT/2000/XP

© 2001 Viewpoint Corporation. All rights reserved.

Special Notes for Macintosh Compatibility

Viewpoint is the first company to integrate the communication between MacOS IE and MacOS IE plug-ins.

To ensure Macintosh compatibility

1. Convert your page to use the MTS3 Interface. (See the section called "Converting Existing Web Pages to Use the MTS3 Interface" for details.)
2. Remove any Mac sniffer scripts from your HTML file. Here's an example of what a Mac sniffer looks like in the JavaScript:

```
var isIE4 = navigator.appName == "Microsoft Internet Explorer" &&
parseInt(navigator.appVersion.substring(0,1)) >= 4;
var agt=navigator.userAgent.toLowerCase();
var isWin = agt.indexOf('win') != -1;
var isMac = agt.indexOf('mac') != -1;
if (isWin) {}
else if (isMac) parent.location="mac_page.html";
```

3. Make sure you have an index.html file in the same folder as your page. It doesn't matter what the index.html contains, so long as it is a valid .html document (that is, it has at least the <html> and </html> tags).

To specify alternate content for ZoomView, Hyperview, or Flash content on a Mac

1. Include a "function altcontent()" line in the HEAD section of your HTML file. For example:

```
<script language="javascript">
  function altcontent() {
    document.write('<img src="flag.jpg" width="633" height="500">');
  }
</script>
```

2. Use JavaScript to check for the MacOS. For example, you could use the following code:

```
<script language="javascript">
      if (isMac) altcontent();
      else vmp = new MTSPlugin("zoomview/flag.mtx", 633, 500, "", "popUp",
"GenieMinimumVersion=50333696");
   </script>
```

This is an example of a Web page that specifies alternate content for MacOS.

```
<html>
 <head>
  <script language="javascript" src="resources/MTS3Interface.js"></script>
  <script language="VBScript"   src="resources/mtsAxDetect.vbs"></script>
  <script language="javascript">
  <!--
    var vmp;    //This global variable is for the VET content

    function altcontent() {
      document.write('<img src="flag.jpg" width="633" height="500">');
  }
  //-->
  </script>
</head>
<body bgcolor=#484747 link=#484747 vlink=#484747 alink=#484747>
  <!-- FIRST OBJECT/EMBED -->
  <script language="javascript">
    if (isMac) altcontent();
    else vmp = new MTSPlugin("zoomview/flag.mtx", 633, 500, "", "popUp",
      "GenieMinimumVersion=50333696");
  </script>
</body>
</html>
```

© 2001 Viewpoint Corporation. All rights reserved.

Calling the MTS3 Interface From Your Web Page

A component of the MTS3 Interface called the **MTS3 Interface constructor** automatically readies your Web page (.html or .htm) for display on on either Microsoft Internet Explorer or Netscape and on either Windows or MacOS.

The constructor recognizes the current browser and the operating system to properly embed the tag and ensure communication between JavaScript function calls and VMP.

Instead of embedding a VET scene and its parameters into a Web page, the MTS3 Interface constructor builds an embed tag when it is invoked. The constructor is called during HTML parsing, meaning that a VET scene cannot display on a Web page after the browser has finished loading the page. However, to accommodate dynamic control of content, you can use multiple layers, one for each VET scene. (When creating layers, be sure to use HTML tags appropriate for each browser you intend to support. See an HTML reference to see which tags are supported by each browser.)

Important: The type of rich media you include in your Web page dictates which required VET components you must list when calling the MTS Interface constructor. See "Required Settings for Your Page's VET Content" for details.

Setting Up VMP Auto-Installation From Your Web Page

The MTS3 Interface checks to see if VMP is installed on the user's computer. If it is not, VMP is installed automatically. This section describes how to set up VMP auto-installation for your Web page.

To set up VMP auto-installation from a Web page

1 If you are installing from a Web page with VET content, do the following:

- Use the latest version of the MTS3 Interface. (If your version of MTS3Interface.js is 3.0.7.33 or earlier, there will not be an automatic refresh of the VET content page. Newer versions will do this automatically. The user will be notified that they need to manually refresh the content page should redirector encounter an old version of MTS3Interface.
- Use the "popUp" installation style option. (See "Syntax Parameters" section for details.)

–Or–

If you are installing from a link (no VET content), open a 470x370 (wide/tall), non-resizable window with no variable names, no toolbars or location bar, using this Web address:
http://www.viewpoint.com/developerzone/download/redirector.html

You can use the following code if you like:

```
function popup() {
      destination =
        "http://www.viewpoint.com/developerzone/download/redirector.html";
      window.open(destination, "vet_installer",
        "width=470,height=370,toolbar=no,location=no,resizable=no");
}
```

2 Specify the content type (ContentType):

- For a 3D installation, use "ContentType=1".
  (If you omit the ContentType variable, the Redirector runs the 3D installation.)
- For a ZoomView installation, use "ContentType=2".

Note: You must either use MTS3Interface version 4.0.x (or are manually start the installation) for this step to work.

© 2001 Viewpoint Corporation. All rights reserved.

MTS Interface versions 3.0.7.33 and earlier don't pass the ContentType to redirector. If these outdated versions encounter a ContentType variable, the variable is ignored and Redirector defaults to a 3D installation.

Example:

```
<script language="javascript">
   vmp = new MTSPlugin("my.mtx", 250, 250, "bkey.txt", "popup",
"ContentType=2");
</script>
```

- If you are using a manual popup, append "?1" for 3D or "?2" for ZoomView to the Web address in your popup: For example:
  `http://www.viewpoint.com/developerzone/download/redirector.html?2`

Creating Non-Layered Scenes

To create non-layered scenes

1 Include the a <script> tag listing the JavaScript library (MTS3Interface.js) in within the HEAD tag of the HTML file, as follows:

```
<script language="javascript" src="MTS3Interface.js"></script>
```

2 Be sure to declare the object variables within JavaScript tags before calling the constructor. (The variable declaration can appear in either the HEAD or BODY of the HTML file.) For example:

```
<script language="JavaScript">
var vmp, vmp2; // Declaration of the object(s)
</script>
```

3 After object variable(s) have been declared, the constructor can be called in the specified HTML location(s). Use the following syntax. (For a definition of these parameters, see the "Syntax Parameters" section below.)

```
<div id="Layer1">
<script language="JavaScript">
vmp = new MTSPlugin(mtx_file, width, height, bkey, "popUp", "layer=Layer1");
…
vmp2 = new MTSPlugin(mtx_file, width, height, bkey, "popUp", "layer=Layer1");
</script>
</div>
```

4 Add any optional parameters, depending on the type of content that you've included in your VET scene. See "Required Settings for Your Page's VET Content" for details.

```
<script language="JavaScript">
vmp = new MTSPlugin(mtx_file, width, height, bkey, "popUp");
…
vmp2 = new MTSPlugin(mtx_file, width, height, bkey, "popUp");
</script>
```

Creating Layered Scenes

To call the MTS3 Interface constructor from inside a layer

1 Follow steps 1 and 2 above.

2 After object variables have been declared, the constructor can be called in the specified HTML location(s). Use the following syntax. (For a definition of these parameters, see following section, "Syntax Parameters".)

```
<script language="JavaScript">
  vmp = new MTSPlugin(mtx_file, width, height, bkey, "popup", "layer=Layer1");
  ...
```

© 2001 Viewpoint Corporation. All rights reserved.

```
  vmp2 = new MTSPlugin(mtx_file, width, height, bkey, “popup”,
    "layer=Layer1");
</script>
```

Note that this parameter syntax is the same as above with the addition of a layer parameter. This parameter allows the constructor to accept the layer name in which a VET scene is to appear and “Layer1” is the name of the actual (and existing) layer.

3 Add any optional parameters, depending on the type of content that you’ve included in your VET scene. See “Required Settings for Your Page’s VET Content” for details.

© 2001 Viewpoint Corporation. All rights reserved.

Syntax Parameters

Parameters	Description
mtx/mtz_file	Path for the .mtx or .mtz file. Example: `vmp = new MTSPlugin("my.mtx", width, height, bkey, "popUp");`

width	Width of the VMP window (pixels or percentage). Examples:
	`vmp = new MTSPlugin(mtx_file, 400, height, bkey, "popUp");`
	
	`vmp = new MTSPlugin(mtx_file, 75%, height, bkey, "popUp");`

height	Height of the VMP window (pixels or percentage). Examples:
	`vmp = new MTSPlugin(mtx_file, width, 400, bkey, "popUp");`
	
	`vmp = new MTSPlugin(mtx_file, width, 75%, bkey, "popUp");`

bkey	Broadcast Key filename/path. Example: `vmp = new MTSPlugin(mtx_file, width, height, "bkey.mtx", "popUp");`
Installation style	There are several ways to handle the installation process when the user does not have VMP installed. You can use the following predefined keywords (or specific HTML code — also called alternative content): Alternate content — Alternate tag. If plug-in detection fails, this argument is used by the constructor to display alternate content, such as a JPEG or GIF image. Example: `"<img src='image.gif' border='0'>"` **Note:** See "Optional parameters" for using "imagelink" and "popup" with alternate content. "classic" — If VMP is not installed, the MTS3 Interface will automatically connect to the Viewpoint component server and prompt the user to start installation of VMP. **Note:** On MacOS, passing "classic" will result in the same installation process as passing "popUp". "none" — Deprecated parameter, replaced by "classic". "popUp" — If the VET plug-in is not installed a popup window will appear with directions on how to download and install the plugin. The MTS3 Interface will detect the browser name/version and the operating system to direct the user to the proper (hard-coded) plug-in download page. Once VMP is installed, the installation page will attempt to automatically refresh the page it was called from, ensuring that VET content is displayed.
Optional parameters	This can include one or more "tokens", separated by semicolons. Each token has a name and a value. This is an example of a constructor with an optional (multi-token) parameter: `vmp = new MTSPlugin(mtx_file, width, height, bkey, "popUp", "GenieMinimumVersion=123456; ComponentMinimumVersion=123456;");` GenieMinimumVersion — If GenieMinimumVersion is not supplied, the interface will use the default value. Example: `vmp = new MTSPlugin(mtx_file, width, height, bkey, "popUp", "GenieMinimumVersion=123456;");` ComponentMinimumVersion — If ComponentMinimumVersion is not supplied, the interface will use

© 2001 Viewpoint Corporation. All rights reserved.

	the default value. Example: `vmp = new MTSPlugin(mtx_file, width, height, bkey, "popUp", "ComponentMinimumVersion=123456;");`
HostMinimumVersion	If HostMinimumVersion is not supplied, the interface will use the default value. Example: `vmp = new MTSPlugin(mtx_file, width, height, bkey, "popUp", "HostMinimumVersion=123456;");`
Layer	If the plug-in object is to appear on a layer, the layer name must be provided via this parameter argument. For example: `vmp = new MTSPlugin(mtx_file, width, height, bkey, "popUp", "Layer=MyLayer;");`

ImageLink	Set this to 1 for alternate content to serve as a link. The interface will automatically wrap the alternate content with a necessary <a href> tag. Once the link is clicked, the Web page will reload and a regular object embed tag will take the place of the content, thereby triggering the plug-in download.
	Note: On MacOS, a popup window will appear instead of inserting an object embed. These are two examples:
	`vmp = new MTSPlugin(mtx_file, width, height, bkey, "<img src='img.gif'>", "ImageLink=1;");`
	
	`vmp = new MTSPlugin(mtx_file, width, height, bkey, "Click to download the VET Plugin", "ImageLink=1;");`

popup	You can use this in conjunction with the ImageLink argument. If ImageLink is set to 1, and popUp is set to 1, then once the link is clicked, a popUp install window will appear instead of inserting an object embed. These are two examples:
	`vmp = new MTSPlugin(mtx_file, width, height, bkey, "<img src='img.gif'>", "ImageLink=1;");`
	
	`vmp = new MTSPlugin(mtx_file, width, height, bkey, "Click to download the VET Plugin", "ImageLink=1;");`

RequiredVersions	If RequiredVersions is not supplied, the interface will use the default value. Example: `vmp = new MTSPlugin(mtx_file, width, height, bkey, "popUp", "RequiredVersions=SreeD.dll=x.x.x.x; SreeDMMX.dll=x.x.x.x;");`
ContentType	Specifies whether the installation is for 3D content (“ContentType=1”)or for ZoomView Content (“ContentType=2”). Example: `vmp = new MTSPlugin(mtx_file, width, height, bkey, "popUp", "ContentType=1;");`

© 2001 Viewpoint Corporation. All rights reserved.

Required Settings For Your Page's VET Content

The type of rich media content you include in your Web page dictates the components you must specify in your constructor call. The following table lists the required components for each rich media content type. (See the "Syntax Parameters" section above.)

For this content...	These components are required
Poser	ComponentMinimumVersion=50332936 GenieMinimumVersion = 50332496
Cursors	HostMinimumVersion=50333440 ComponentMinimumVersion=50333440
ZoomView	ComponentMinimumVersion=50333440 **Note:** Most ZoomView content also uses pickable hot spots (widgets), which requires the following: HostMinimumVersion=50333440 GenieMinimumVersion = 50333440 RequiredVersions=SreeD.dll=3.0.7.0,SreeDMMX.dll=3.0.7.0
HyperView	ComponentMinimumVersion=50333696 **Note:** HyperView also requires <MTSScene Version="308" />
SWFView	ComponentMinimumVersion=50333440
Pickable hot spots (widgets)	HostMinimumVersion=50333440 GenieMinimumVersion = 50333440 RequiredVersions=SreeD.dll=3.0.7.0,SreeDMMX.dll=3.0.7.0
Specular wrap	HostMinimumVersion=50333440 GenieMinimumVersion = 50333440 RequiredVersions=SreeD.dll=3.0.7.0,SreeDMMX.dll=3.0.7.0
Complex interactions (anything more than OnClick)	ComponentMinimumVersion=50333440

© 2001 Viewpoint Corporation. All rights reserved.

Converting Existing Web Pages to Use the MTS3 Interface

You can quickly update any existing Web pages (.html) that use the object/embed tag to use the MTS3Interface.js file and the constructor tag. The steps below will take you through the process.

To update an existing HTML page to use the MTS3 Interface and constructor tag

1. Copy the following file in the folder with your content:
   - MTS3Interface.js
2. Include references to the object variables and JavaScript library (MTS3Interface.js) in within the HEAD tag of the HTML document:

```
<script language="javascript" src="MTS3Interface.js"></script>
<script language="JavaScript">
var vmp; // Declaration of the object
</script>
```

3. Replace your standard Viewpoint Object/Embed with the following MTS3 Interface constructor tag:
   - For non-layered pages, use a call like this:

```
<script language="javascript">
vmp = new MTSPlugin("filename.mtx", 600, 600, "bkey.txt", "popUp"
</script>
```

   - For layered pages, use a call like the following. (Note that content placed in a layer must reference the layer name in the variable call.)

```
<div id="Layer1" style="position:absolute; left:20px; top:350px;
width:310px; height:310px; z-index:1">

<script language="javascript">
vmp = new MTSPlugin("ase_pub_alpha.mtx", 300, 300, "bkey", "popUp",
"layer=Layer1");
</script>

</div>
```

4. Change JavaScript calls between the HTML page and VMP with the new MTS3 Interface JavaScript functions, such as the following:

```
onclick="vmp.TriggerAnim('name of animation')"
```

   For example, change this: `onclick="triggeranimation('anim1')`

   To this: `onclick="vmp.TriggerAnim('anim1')`

5. Remove all previous functions which are now unused from the HTML.

© 2001 Viewpoint Corporation. All rights reserved.

Calling JavaScript Functions for the MTS3 Interface

After a successful constructor call, a JavaScript object becomes available for various state manipulations. Just as you could previously call JavaScript Viewpoint Media Player (VMP) plug-in functions, you can call similar functions of the newly created JavaScript object that, in turn, can communicate with VMP.

JavaScript object functions can be called as follows:

```
<script language="JavaScript">
...
vmp.ClearScene();
...
</script>
```

JavaScript Object Functions

This table describes the available JavaScript object functions:

Function Name	Function Description
ClearScene()	Clears the object scene. Example: `Cube1.ClearScene();`
GetProperty(name, prop, value, type)	Gets object property. Example: `Cube1.GetProperty('MTSInstance.RotRoot', 'rot_', value, 'mts_pnt3d');`
GetVer(comp)	Gets component version. The function will return a version number of the specified component.Example: `Cube1.GetVer('ISceneComponent');`
LoadMTX(path)	Loads the .mtx file. Example: `Cube1.LoadMTX("anotherCube.mtx");`
Render()	Renders the scene. Example: `Cube1.Render();`
ResetAnim(anim)	Resets object animation. Example: `Cube1.ResetAnim ('Rotate');`
ResetCamera()	Resets the object camera. Example: `Cube1.ResetCamera();`
ReverseAnim(anim)	Reverses object animation. Example: `Cube1.ReverseAnim ('Rotate');`
SetCollapsed(name, value)	Sets collapsed. Example: `Cube1.SetCollapsed('MTSInstance.RotRoot', 1);`
SetProperty(name, prop, value, type)	Sets object property. Example: `Cube1.SetProperty('MTSInstance.RotRoot', 'rot_', value, 'mts_pnt3d');`
SetVisible(name, value)	Sets object visibility. Example: `Cube1.SetVisible('MTSInstance.RotRoot', 1);`
StartAnim(anim)	Starts object animation. Example: `Cube1.StartAnim('Rotate');`
StopAnim(anim)	Stops object animation. Example: `Cube1.StopAnim('Rotate');`
ToggleCollapse(name)	Toggles collapse. Example: `Cube1.ToggleCollapse('MTSInstance.RotRoot');`
TogglePano(value)	Toggles panorama. Example: `Cube1.TogglePano(value);`
ToggleVisible(name)	Toggles visibility. Example: `Cube1.ToggleVisible('MTSInstance.RotRoot');`
TriggerAnim(anim)	Triggers object animation. Example: `Cube1.TriggerAnim ('Rotate');`
PostEvent(name, delay)	Posts an event/message into the MTX Interactor system. The *name* is a string

© 2001 Viewpoint Corporation. All rights reserved.

	defining the name of the event as handled in the MTX. The *delay* is either 0, or a number of seconds to wait before the message is actually posted into the system. The delay occurs inside the MTX, not in JavaScript.
GetLastErrCode()	Returns the last error reported by VMP jscript handler; for example, from SetProperty or GetProperty. Codes returned (integer): No Error = 0 Animation Commandd Failed= 1 Set Object Property Failed= 2 Get Object Property Failed= 3 LoadMTX Failed= 4 TrackData Failed= 5 Upload DataTrack Failed= 6 Toggle Failed = 7 Render Failed= 8 ClearScene Failed = 9 File Path Not Found= 10 Invalid Param= 11 No Loader Avail= 12 Object Not Valid= 13 Add Event Failed= 14 Root Instance Failed= 15 DataTrak SetProps Failed= 16 VMP Error Reporter Failed= 17 Any UnKnown Error = 1000
GetLastPluginErr(type)	Returns the last error reported by VMP and its scene. Currently handles only type = "BroadcastKey". Errors: GE_NoError = 0 GE_Unknown = 2000 Errors for type = "BroadcastKey": GE_BKeyExpired = 1001 GE_BKeyFeatureBit = 1002 GE_BKeyDownloadFailed = 1003 GE_BKeyBadHash = 1004 GE_BKeyBadURL = 1005 GE_BKeyOldStyle = 1006 GE_BKeyBadFormat = 1007
SetPluginErr(type, value)	Can set an error to VMP, where *type* is the category and *value* is the error code.
GetAllPluginErrFor(type)	Returns the all errors reported by VMP and its scene for the type (category).
ClearAllPluginErrs(type)	Clears all VMP errors from a category.

Other JavaScript Functions and Variables

In addition to the object functions above, the MTS3 Interface provides several other useful utilities that are not scene specific. These are listed in the following table:

Function/Variable Name	Function/Variable Description
isIE	Boolean variable within the MTS3 Interface that contains “true” if the browser that is currently displaying the HTML document is Microsoft Internet Explorer. Example: `if (isIE) alert ('MSIE');`
isNN	Boolean variable within the MTS3 Interface that contains “true” if the browser that is currently displaying the HTML document is Netscape. Example: `if (isNN) alert ('Netscape');`
isWin	Boolean variable within the MTS3 Interface that contains “true” if the operating system is currently running Windows. Example: `if (isWin) alert ('Windows');`
isMac	Boolean variable within the MTS3 Interface that contains “true” if the operating system currently running is MacOS. Example: `if (isMac) alert ('MacOS');`
IsMTSInstalled()	A function that returns a boolean value of “true” if the Viewpoint plugin is installed. Example: `if (IsMTSInstalled()) alert ('Viewpoint Installed!');`

To see examples of the object function calls, see the MTS3 Interface pages on the Viewpoint Developer Central Web site (http://developer.viewpoint.com/). Then click **Calling JavaScript Functions for the MTS3 Interface**.

© 2001 Viewpoint Corporation. All rights reserved.

Using the MTS3 Interface Debugger

Viewpoint's MTS3 Interface has a JavaScript debugger feature that facilitates Viewpoint Media Player (VMP) integration and control.

The debugger can also be used as a help tool for JavaScript coding. The debugger automatically displays all JavaScript calls that are made by the MTS3 Interface object.

To see a demo of the MTS Interface debugger, see the MTS3 Interface pages on the Viewpoint Developer Central Web site (http://developer.viewpoint.com/). Then click **Using the MTS3Interface Debugger**.

Enabling the MTS Interface Debugger

To enable the debugger

- Call the MTSDebugger() function as follows:

```
<script language="JavaScript">
...
MTSDebugger(1);
...
</script>
```

Disabling the MTS Interface Debugger

To disable the debugger

- Call the MTSDebugger() function as follows:

```
<script language="JavaScript">
...
MTSDebugger(0);
...
</script>
```

In addition to automatically displaying JavaScript object calls, the debugger window can be used to avoid the cumbersome use of multiple alert statements.

Sending a String to the Debugger Window

To send a string to the debugger window

1. Ensure that the debugger is enabled.
2. Call the MTSConsole() function as follows:

```
<script language="JavaScript">
...
MTSConsole("Some Text or a Primitive Variable");
...
```

© 2001 Viewpoint Corporation. All rights reserved.

Chapter 3:
Controlling XML Animations From JavaScript

Controlling scene animations from the HTML page is a powerful feature of VET Web applications. You can use JavaScript to embed animation triggers and controls in the .html file, allowing you to incorporate scene navigation icons and buttons directly in the Web page design. You can allow users to trigger animations that

- Change the color, material, or texture of a product.
- Show a product from different perspectives.
- Demonstrate the functionality of a product.

By clicking buttons on the bottom of the HTML page, users can see many views of this network server, as well as trigger animations showing functionality.

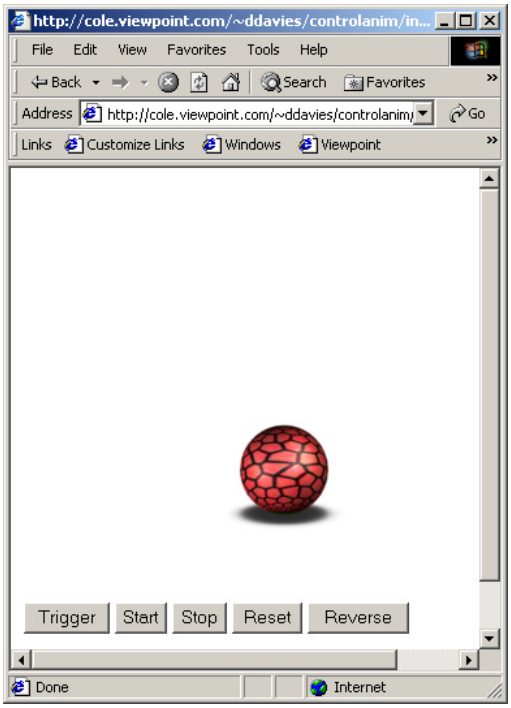
The following examples show different aspects of controlling animations from an HTML page.

© 2001 Viewpoint Corporation. All rights reserved.

Using JavaScript Animation Control Functions in a Scene

Embedding animation controls in the HTML page allows you create product Web pages with consistent look and feel in the navigation. You can create a Web page template with the navigation and buttons you want, and then embed a VET scene in each one. A bit of JavaScript tweaking in each .html file allows you to easily hook up the animations in the .mtx file for each scene to the navigation icons on its Web page.

If you have scripted interactors in an .mtx file, you are likely familiar with the animation controls available as values for Action. (For more information on XML scene interactors, see *Viewpoint Scene Interactors Authoring Guide* on Viewpoint's Developer Central Web site.) With JavaScript, you can set these basic animation controls on the HTML page: trigger, start, stop, reset, and reverse functions.



Live example: http://cole.viewpoint.com/~ddavies/controlanim/index.html

Scene files: http://cole.viewpoint.com/~ddavies/controlanim/index.zip

JavaScript animation controls follow the same syntax: *AnimationControl* ('*animator_name*'). The buttons generated on the HTML page are standard button forms called by JavaScript. Here is an example of a section of a form from the HTML page with buttons that trigger each of the functions:

```
<form>
<input type=button value="Trigger"
onclick="pluginObj.TriggerAnim('move_ANIM')">
<input type=button value="Start"   onclick="pluginObj.StartAnim('move_ANIM')">
<input type=button value="Stop"    onclick="pluginObj.StopAnim('move_ANIM')">
<input type=button value="Reset"   onclick="pluginObj.ResetAnim('move_ANIM')">
<input type=button value="Reverse"
onclick="pluginObj.ReverseAnim('move_ANIM')">
</form>
```

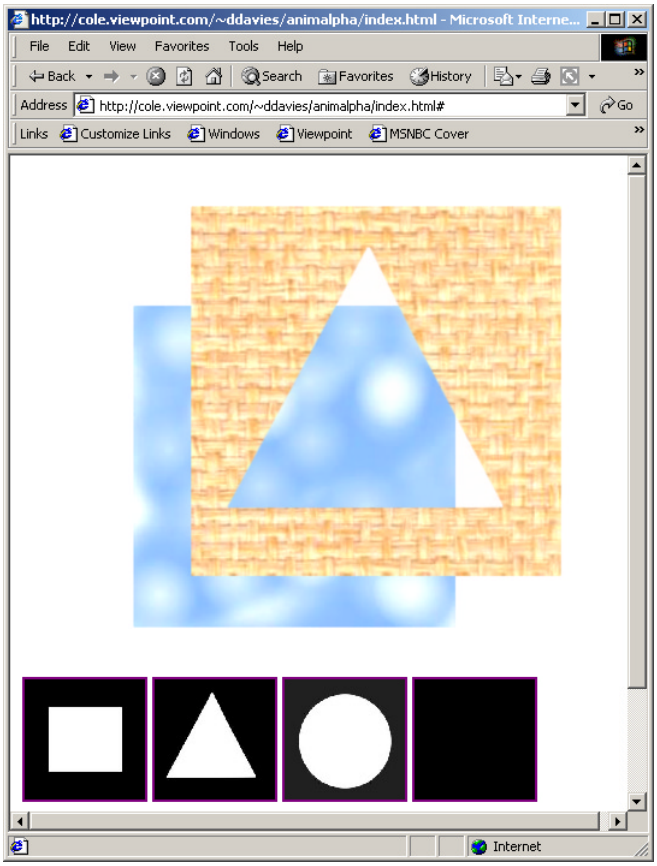
The animation named *move_ANIM* refers to an animation in the scene's .mtx file, under MTSTimeElem.

© 2001 Viewpoint Corporation. All rights reserved.

Triggering XML Animations From Icons on an HTML Page

Like the previous example, this VET Web application has animation triggers on the HTML page. Rather than using the standard button calls (that produce generic gray buttons), you can use custom icons as buttons to create your own user interface. With custom UI graphics, you can give users better visual cues. For instance, if your scene shows different product colors by means of a texture animation, you can create buttons showing the different product colors.

The example scene *AnimAlpha* uses JPEG icons on the HTML page to trigger alpha channel animations scripted in the .mtx file.



Click on each of the black shapes at the bottom of the HTML page to apply a different alpha map to the scene above.

Live example: http://cole.viewpoint.com/~ddavies/animalpha/index.html

Scene files: http://cole.viewpoint.com/~ddavies/animalpha/index.zip

Create an Animation Trigger Call With a Referenced Icon

An OnClick tag is attached to the anchor tag (href) in order to make a trigger animation call. Here's the code in the .html file:

```
<a href="#" OnClick="pluginObj.TriggerAnim('alpha_square')"><img
src="alpha_square.jpg" width=100 height=100></a>
```

Notice how there is no href link on the anchor tag; there's only a pound sign (#) character. Normally, the href value is a link to another HTML page; however, the # character means that no navigation to another page occurs. Instead, the OnClick tag captures that click and allows you to call a JavaScript function. An icon where the user clicks to trigger the animation—alpha_square.jpg—is referenced here as well.

© 2001 Viewpoint Corporation. All rights reserved.

In this example, the OnClick calls a JavaScript function that belongs to the VET scene (pluginObj). This scene was constructed into a variable called *pluginObj*, like this:

```
. . .
<script language="javascript">
   var pluginObj;
</script>
</head>
<body>
<script language="javascript">
   pluginObj = new MTSPlugin("AnimAlpha1.mtx", 500, 400,
"../bkey.txt", "none", "GenieMinimumVersion=50333494;
ComponentMinimumVersion=50333494");
</script>
. . .
```

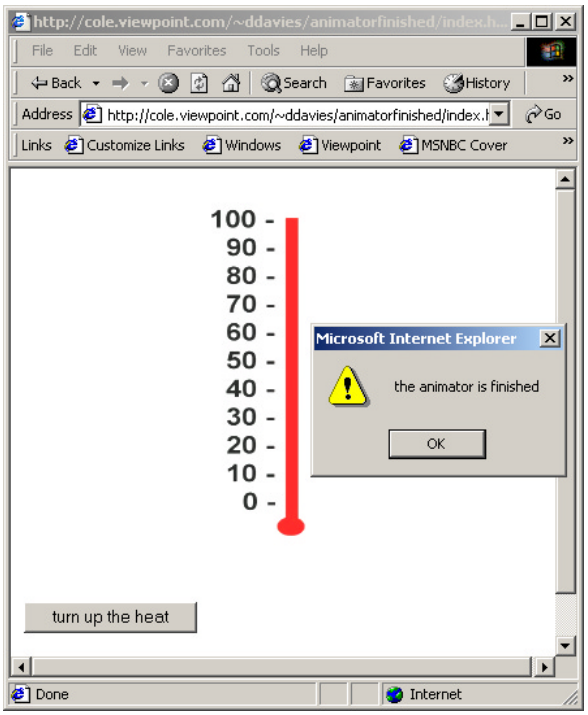
To recap, triggering an animation in the AnimAlpha1.mtx file requires that you identify the instance variable *pluginObj*, followed by a period, the *TriggerAnim* function name, and a quote-delimited animator name within parentheses:

```
pluginObj.TriggerAnim('animator_name_from_mtx')
```

© 2001 Viewpoint Corporation. All rights reserved.

Triggering an XML Animation and Creating an Animator-Finished Alert

JavaScript animation controls can be scripted in the .mtx or .html file. In this example, a JavaScript button on the HTML page triggers an animation in the .mtx file. When the scene animation finishes, a JavaScript call is made from script in the .mtx file to the Web browser triggering an alert that lets the user know the animation has completed.



Live example: http://cole.viewpoint.com/~ddavies/animatorfinished/index.html

Scene files: http://cole.viewpoint.com/~ddavies/animatorfinished/index.zip

The VET object in this scene was constructed into a variable called *a*, like this:

```
<script language="javascript">
<!--
var a;   //This global variable is for the VET content.
//-->
</script>
</head><body>
<script language="javascript">
<!--
   a=new MTSPlugin("thermometer.mtx",400,300,"","popUp","");
//-->
</script>
```

© 2001 Viewpoint Corporation. All rights reserved.

This scene also adds JavaScript to the button in the form. Just as the anchor tag (href) was used for an OnClick in the previous example, it's also possible to use a form button and add an onclick to it. The onclick in this instance utilizes the button onclick event to trigger an animator (inside thermometer.mtx) called *cool_animator*. Here is how the onclick is attached to a button:

```
<form>
   <input type=button value="turn up the heat"
onclick="a.TriggerAnim('cool_animator')">
</form>
```

The animator (*cool_animator*) makes a JavaScript call once it is finished.

JavaScript in the .mtx File

In the .mtx file, you have access to any JavaScript function that you explicitly create and also all of the standard JavaScript functions provided by the browser. One such standard function is the *alert* function. It takes one parameter—a string—and outputs that text string into a pop-up dialog box that requires *OK* to be clicked.

In the .mtx file in this example, the alert function is called displaying the text string "The animator is finished." This section of the scene's .mtx file contains the JavaScript function that generates the alert box:

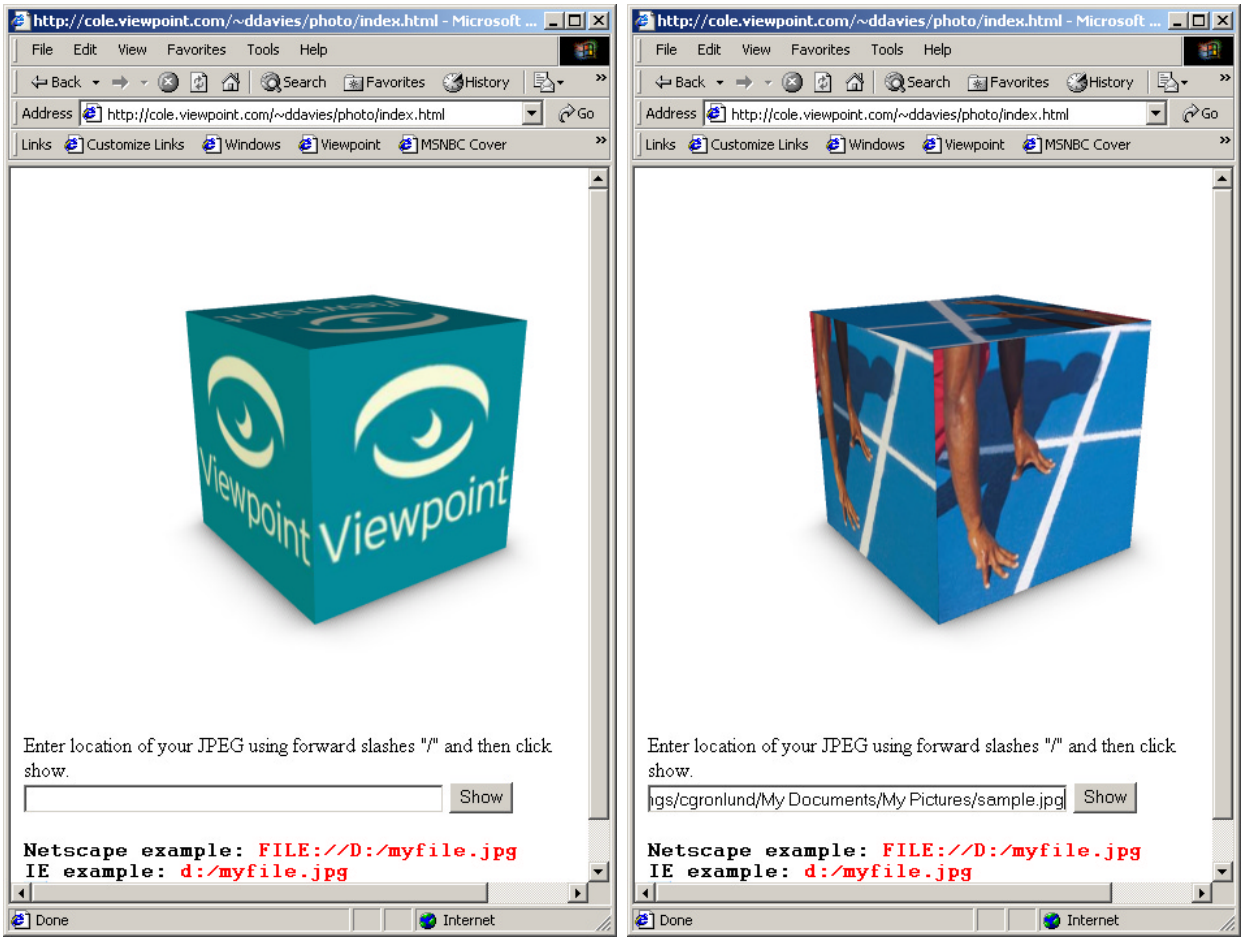
```
<MTSTimeElem Type="Keyframe" Name="cool_animator" On="0" >
   <Target Name="Simple_1" Property="loc_" Timeline="T0"/>
   <Time> 0 1 2 </Time>
   <Timeline Name="T0" Type="3D"> [0.1 0.1 0.1] - +[0 0.6 0] </Timeline>
   <MTSTimeElem Type="ActionAnimator">
      <Trigger Target="dummy"/>       <!-- Time 0 -->
      -                               <!-- Time 1 -->
      <Action="MTSJavaScript" Func="alert('The animator is finished.')"/>
                                      <!-- Time 2 -->
   </MTSTimeElem>
</MTSTimeElem>
```

© 2001 Viewpoint Corporation. All rights reserved.

Creating a Dynamic XML Animator That Applies a User-Selected Texture to a Primitive

In your VET Web applications, you might find it useful to allow users to customize the VET scene or model. For instance, a product Web page for picture frames might allow a user to choose a personal image file to display.

This example shows how you can create a VET Web application in which a user can specify an image to be applied as a texture in a VET scene. In this example, the texture is applied to a primitive cube and can be specified with a URL or file path.



In this simple example scene, the user can specify the name and path of a JPEG in a Web page field. The image is then applied to a primitive cube.

Live example: http://cole.viewpoint.com/~ddavies/photo/index.html

Scene files: http://cole.viewpoint.com/~ddavies/photo/index.zip

JavaScript stores the JPEG path entered into a variable that is inserted into an XML string using standard string concatenation methods. This XML string is then sent to Viewpoint Media Player, and becomes part of the scene, just as if it were coded in the .mtx file.

© 2001 Viewpoint Corporation. All rights reserved.

Here is the JavaScript:

```
<script language="javascript">
<!--
var pluginObj;

function shownewuserpic()
{
   //get the path that the user entered in the form
   var newpath = document.myform.newpath.value;
   var str = ('<MTSTimeElem Type=\\"MTSImageStream\\" Name=\\"streamnewpic\\"
On=\\"1\\" Path=\\"' + newpath + '\\"><Target
Name=\\"MTSTexture.photo_TEXTURE_0\\"/> </MTSTimeElem>');

   pluginObj.Execute(str);
   pluginObj.TriggerAnim('streamnewpic');
   pluginObj.Render();
}
//-->
</script>
</head>
<body>
<script language="JavaScript">
<!--
   pluginObj = new MTSPlugin( "photo.mtx", 500, 400, "../bkey.txt", "none",
"GenieMinimumVersion=50333494; ComponentMinimumVersion=50333494" );
//-->
</script>
<form name="myform" onsubmit="shownewuserpic();return false">

   Enter JPEG location using forward slashes "/" and then click show.<br>

<input type="text" name="newpath" size="50">
   <input name="trgr" type=button value="Show" onclick="shownewuserpic()">
<br>
</form>
<font face="Courier"><b>
Netscape example:  <font color="red">FILE://D:/myfile.jpg</font><br>
IE example: <font color="red">d:/myfile.jpg</font><br>
Or Web: <font color=”red”>http://someplace.com/somefile.jpg</font><br>
</b></font>
```

Following is the XML code for this scene:

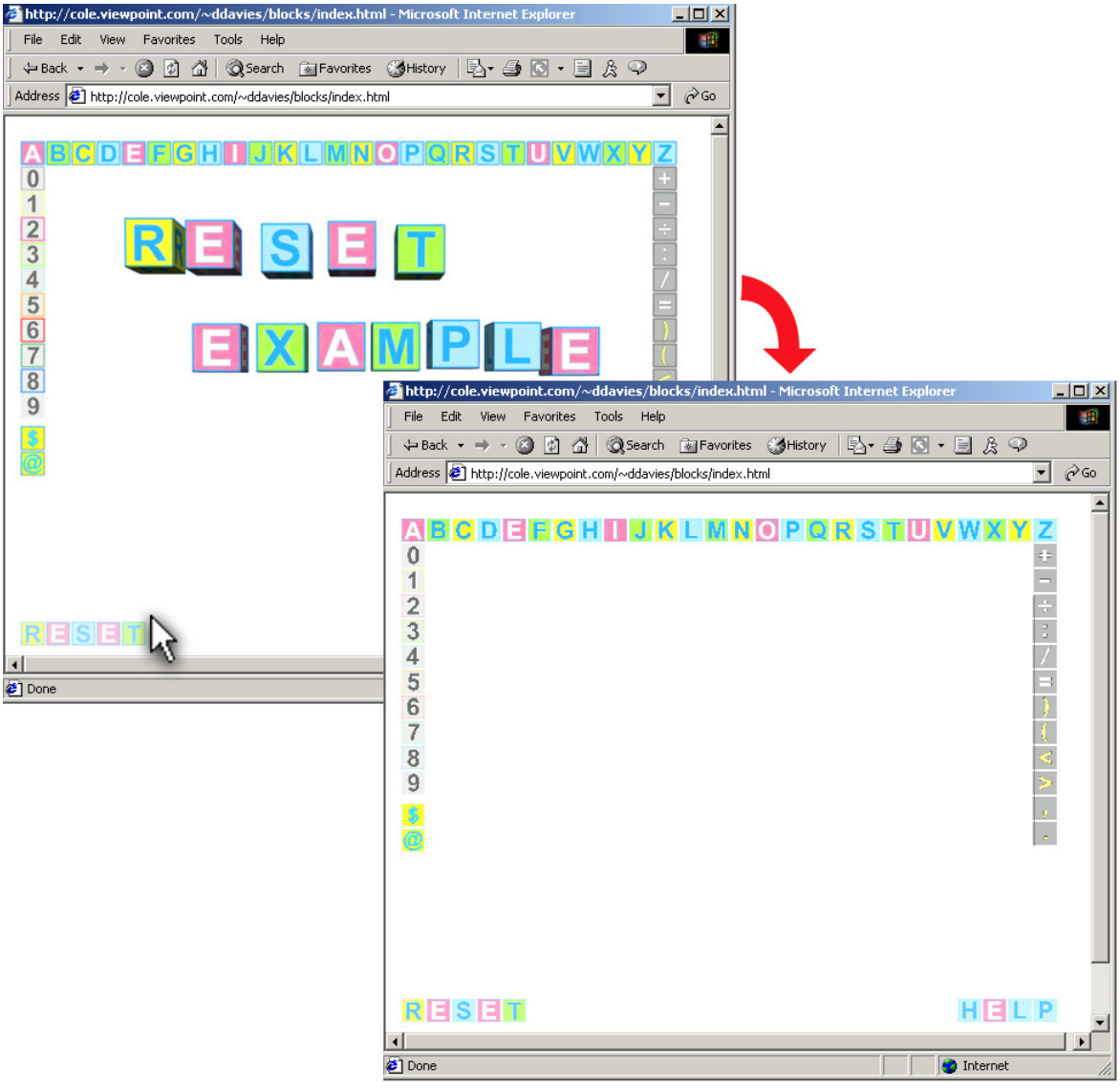
```
<MTSInstance Name="Simple_0" AntiAlias="0">
   <MTSGeometry Name="MTSSimple_0" Type="MTSCube"/>
   <MTSMaterial Name="photo_MATERIAL_0" ID="0" RenderMode="Default" >
      <MTSTextureMap Type="Diffuse" Name="photo_TEXTURE_0"/>
      <MTSColor Type="Diffuse" r="0.65" g="0.65" b="0.65"/>
   </MTSMaterial>
</MTSInstance>
<MTSTimeElem Type="MTSImageStream" On="1" Path="viewpoint.jpg">
   <Target Name="MTSTexture.photo_TEXTURE_0"/>
</MTSTimeElem>
```

© 2001 Viewpoint Corporation. All rights reserved.

Chapter 4: Refresh, Reload, and Resize a Scene From JavaScript

Refreshing (Reloading) the VET Web Application Using JavaScript

While orchestrating how your customers experience your product online, you also want your customers to have maximum interaction with your product. With a VET Web application, they can rotate, zoom in on, and move the object for a better look. These interactions may distort the product's appearance, so you may also want to allow users to reset the VET scene to its original state with a button on the HTML page.



The Reset button on the lower left causes the scene to reload, which, in effect, resets all animations and manual scene manipulations and shows the scene in its original streamed state.

Live example: http://cole.viewpoint.com/~ddavies/blocks/index.html

Scene files: http://cole.viewpoint.com/~ddavies/blocks/index.zip

© 2001 Viewpoint Corporation. All rights reserved.

The JavaScript function as defined here enables the VET Web application to be reloaded on demand. (The link above is to a much larger scene, but the JavaScript and XML shown below are found within that scene.)

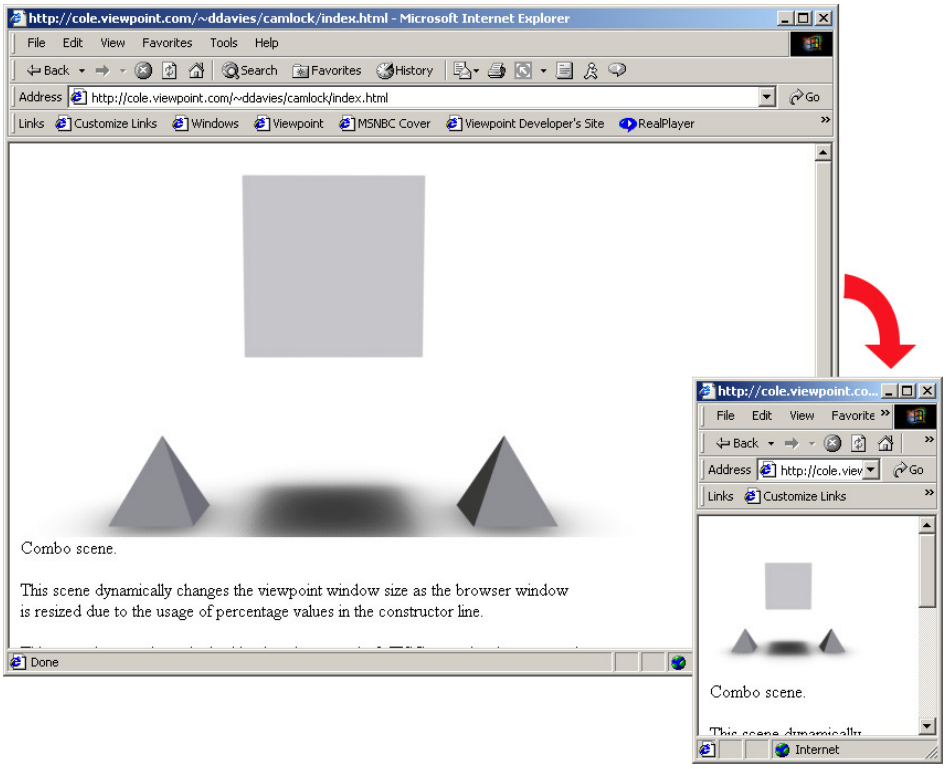
```
<script language="JavaScript">
<!--
function ResetClicked()
{
location.reload()
}
//-->
</script>
```

In the .mtx file, the function ResetClicked is called like this:

```
<OnClick Action="MTSJavaScript" Target="ResetClicked()" />
```

Automatically Resizing the VET Scene When the Browser Window Resizes

As you can control the size of the Web browser window on a user's desktop, you also can control whether the VET Web application appears complete within it or is cropped. This example shows how you can set the VET scene to resize when the browser window is resized.



Live example: http://cole.viewpoint.com/~ddavies/camlock/index.html

Scene files: http://cole.viewpoint.com/~ddavies/camlock/index.zip

© 2001 Viewpoint Corporation. All rights reserved.

Standard Scene Constructor With a Fixed Size

When you embed a VET scene in a Web page, the standard scene constructor line defines the scene window as being an absolute size in pixels. (The pixel values don't need to be within quotation marks, because they're absolute values.)

```
<script language="javascript">
   vmp = new MTSPlugin("widgettest.mtx", 400, 400, "", "popUp",
"GenieMinimumVersion=50332496");
</script>
```

Scene Constructor With Dynamic Scene Size

By using percentage values for the scene dimensions, the scene size becomes a dynamic and its dimensions are values relative to the browser window size. As the window is resized, the VET window is resized, too, according to the percentage specified. Because the % character is used, the numbers are no longer integers and must appear within quotation marks. (This is standard within HTML and is true with tables, also.)

Here is a VET scene constructor with dynamic values:

```
<script language="javascript">
   a = new MTSPlugin("test.mtx", "80%", "80%", "../bkey.txt",
"classic", "GenieMinimumVersion=50333494");
</script>
```

© 2001 Viewpoint Corporation. All rights reserved.

Chapter 5: Setting and Constraining Scene and Object Parameters With JavaScript

Restricting the Movement of a Scene Object

By default in a VET Web application, the user can move and rotate around an object along any and all of the 3D axes, x, y, and z. With JavaScript, you can restrict the axis along which an object can be moved.

Note: This functionality is not available in XML.

Live example: http://cole.viewpoint.com/~ddavies/restrictdrag/index.html

Scene files: http://cole.viewpoint.com/~ddavies/restrictdrag/index.zip

In this example, the user chooses from radio buttons on the HTML page the axis to which movement is restricted. In scripting this, you are creating an illusion: When the user drags the cube around the screen, what actually is happening is that an invisible cube containing the visible cube is capturing the drag action. When you drag the invisible cube, its new position is sent to JavaScript, which uses that information to determine the position of the visible cube. To see the invisible cube and get a feel for what's actually happening, you can modify the .mtx file to change the opacity of the invisible cube to 0.1 (instead of 0.0).

Because you want to restrict the visible cube's movement to one axis, you don't want the visible cube to follow the invisible cube around along all axes, only along one. To accomplish this, the JavaScript dragged( ) function ignores two of the new coordinates and uses the new coordinate for just the selected axis to set the position of the visible cube. If the x axis is selected, then each time the JavaScript receives a new position for the invisible cube, it ignores the new y and z values, keeping the fixed position values for the visible cube instead. It takes the new value for the x axis and uses it with the fixed y and z values in the SetProperty call on the visible cube.

Finally, the dragged function sets the position of the invisible cube to be the same as the visible cube to prevent them from separating.

Note: Netscape Navigator has limited ability to handle the volume of events required in this example.

Here is the JavaScript for this example:

```
<script language="JavaScript">
<!--
var a;
function dragged()
{
   //Get the new coordinates of the invisible cube that
   //has been dragged.
   var coords =
String(a.GetProperty('MTSInstance.invisible_larger_cube','loc_','mts_pnt3d')).
split(" ")
   var x = coords[0]
   var y = coords[1]
   var z = coords[2]
   var newpos = coords

   //Now decide which of the new coordinates you're going to keep and which
   //you're going to set back to their default value, zero (0). If the
   //current restriction is to drag along X and the a new coordinate is
   //x=1, y=1 and z=1, keep the new X value of 1 and set Y and Z back to 0
   //(the default value).
```

© 2001 Viewpoint Corporation. All rights reserved.

```
    if (document.myform.restrict[0].checked == true)  //Restricting to X ?
    {
       var newpos = x + ' 0 0'
    }
    else if (document.myform.restrict[1].checked == true)  //Restricting to Y ?
    {
       var newpos = '0 ' + y + ' 0'
    }
    else if (document.myform.restrict[2].checked == true)  //Restricting to Z ?
    {
       var newpos = '0 0 ' + z
    }
    else
    {
       //None of the radio buttons is selected,
       //so leave newpos == coords. No restriction on drag.
    }
    a.SetProperty('MTSInstance.CubeA0','loc_',newpos,'mts_pnt3d')
    //now re-sync the invisible cube back to where the visible one is
    //to avoid potential separation during quick dragging.
    a.SetProperty('MTSInstance.invisible_larger_cube','loc_',newpos,'mts_pnt3d'
)
    a.Render()
}

//-->
</script>
</head>
<body>
<script language="javascript">
<!--
    a = new MTSPlugin("Blocks.mtx", 600, 450, "../bkey.txt", "none",
    "GenieMinimumVersion=50333494; ComponentMinimumVersion=50333494");
//-->
</script>
<br>This scene restricts the drag movement to only along the selected co-
ordinate.<br>
<form name="myform".
<br>Restrict drag to
    <input type=radio name="restrict" value="1"  CHECKED   >X  
    <input type=radio name="restrict" value="2"         >Y  
    <input type=radio name="restrict" value="3"         >Z
</form>
```

© 2001 Viewpoint Corporation. All rights reserved.

Here is the XML side of this process in the .mtx file:

```
<MTSInstance Name="Blocks">
   <MTSInstance Name="CubeA0">
      <MTSGeometry Name="MTSSimple_A0" Type="MTSCube"/>
      <MTSMaterial Name="Blocks_MATERIAL_0" ID="0" />
   </MTSInstance>

   <MTSInstance Name="invisible_larger_cube" Opacity="0">
      <Transform>
         <Scale x="1.01" y="1.01" z="1.01"/>
         <Position x="2" y="0" z="0"/>
      </Transform>

      <MTSGeometry Name="MTSSimple_Invisible" Type="MTSCube"/>

         <MTSHandle Event="MouseDrag" Action="MTSDrag"
Target="invisible_larger_cube" />

         <MTSHandle Event="MouseDrag" Action="MTSJavaScript"
Target="dragged()" />

      <MTSMaterial Name="Blocks_MATERIAL_1" ID="0" />
   </MTSInstance>
</MTSInstance>
```

In the XML, there is no scale tag for the instance CubeA0 (the visible cube), so the default scale (x="1" y="1" z="1") is applied. For the invisible cube, the scale is 1.01,1.01,1.01 to make it slightly larger than the visible cube.

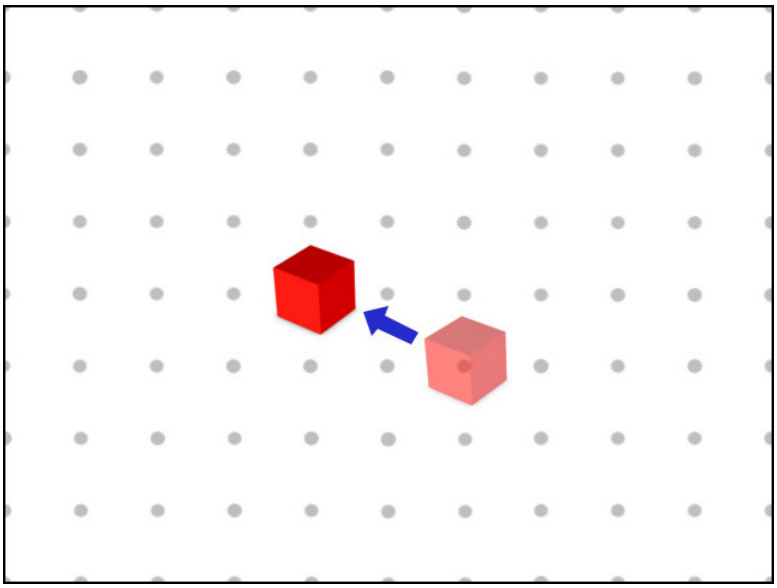
The MouseDrag event is handled twice: The first event handler moves the invisible cube itself. (If that weren't there, then dragging the mouse over the cube would cause the camera to rotate.) The second event handler calls the JavaScript function dragged( ) to inform it that the invisible cube has moved.

Note: In this example, the new position is not passed to JavaScript. The JavaScript function should inquire where the invisible cube has moved to using the GetProperty function call.

© 2001 Viewpoint Corporation. All rights reserved.

Restricting an Object's Movement to Positions on a Grid

By restricting movement of an object to points on a grid, you are creating a "snap to grid" effect as used in many graphics applications. In this example, as you drag the cube, it moves only to the nearest fixed positions in a grid.



Live example: http://cole.viewpoint.com/~ddavies/snap/index.html

Scene files: http://cole.viewpoint.com/~ddavies/snap/index.zip

Here you use the same principle as the previous example that restricts the dragging of a cube by use of an invisible larger cube (see Restricting the Movement of a Scene Object). In this case, all three coordinates of the invisible cube are used, but are rounded to grid values. First, the three coordinates of the invisible cube (which has been moved to a new position) are converted from real numbers to integers using the parseInt function. Then, they are converted back from integers to real numbers and used in a SetProperty for the position of the visible cube.

The result of this two-part conversion is that the new coordinates of the invisible cube as it's being dragged don't have any effect on the visible cube until they increment to the next integer value.

For example, let's take the x coordinate of the cubes. The visible cube has an x position value of 1.000000 and the invisible cube is the same. As the invisible cube is dragged so that its x position increases, it passes through 1.1, 1.2, 1.3 all the way up to 1.9999999. None of these values has any effect on the visible cube, because converting 1.1 or 1.999999 to an integer returns the value 1. Converting the integer value 1 back to real makes it 1.00000, so if you use that in a SetProperty call on the visible cube it naturally doesn't move because it's already at position 1.000000.

Once the position of the invisible cube passes 2.000000 all the way up to 2.999999, it is converted to 2.000000 when it is converted to integer and back. So, after 2.000000 is passed by the invisible cube, all future SetProperties on the visible cube cause it to move to x position 2.000000. This process is used on the x, y, and z coordinates to implement a 3D snap grid.

Note: Unlike the previous example, this scene doesn't need to inquire of the new position of the invisible cube. The new location is sent in the JavaScript function call to the dragged( ) function. This scene also has a second function called stop_drag( ) called by the .mtx file when the user clicks or releases the left mouse button. The stop_drag function moves the invisible cube back to the location of the visible cube to prevent their being separated. This is simply a different way to implement the same thing.

© 2001 Viewpoint Corporation. All rights reserved.

Here is the JavaScript code for this example:

```
<script language="JavaScript">
<!--
var a;

function dragged(loc)
{
   //Get the new coordinates of the invisible cube now
//that it's been dragged somewhere else

   var coords = String(loc).split(" ")  //split coordinates into an array.
   var newpos = ' ' + parseInt(coords[0]) + ' ' + parseInt(coords[1]) + ' ' +
parseInt(coords[2]);

   a.SetProperty('MTSInstance.CubeA0','loc_',newpos,'mts_pnt3d')
   a.Render()
}

function stopdrag()
{
   var coords =
String(a.GetProperty('MTSInstance.CubeA0','loc_','mts_pnt3d')).split(" ")
   a.SetProperty('MTSInstance.invisible_larger_cube','loc_',coords,'mts_pnt3d'
)
   a.Render()
}

//-->
</script>
</head>
<body>
<script language="javascript">
<!--
   a = new MTSPlugin("Blocks.mtx", 600, 400, "../bkey.txt", "none",
"GenieMinimumVersion=50333494; ComponentMinimumVersion=50333494");
//-->
</script>
```

© 2001 Viewpoint Corporation. All rights reserved.

Following is the XML for this scene:

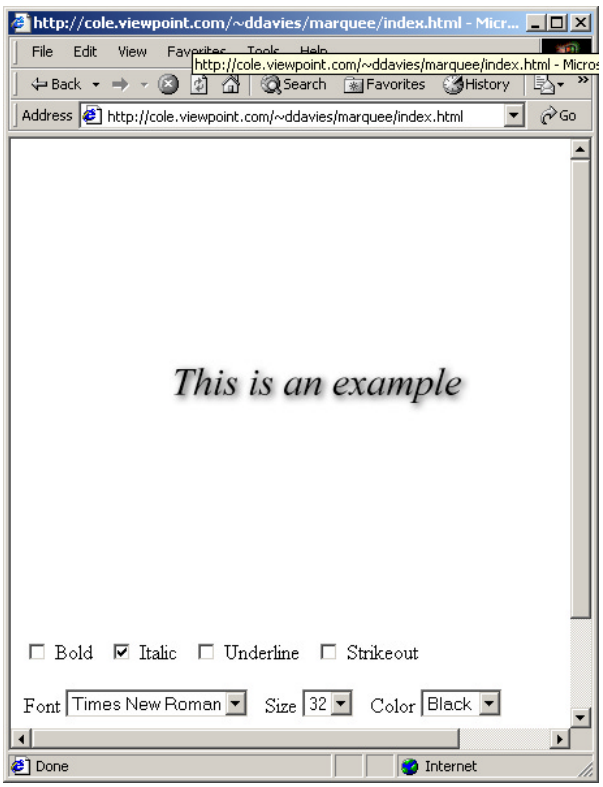
```
<MTSInstance Name="Blocks">
   <MTSInstance Name="CubeA0">
      <MTSGeometry Name="MTSSimple_A0" Type="MTSCube"/>
      <MTSMaterial Name="Blocks_MATERIAL_0" ID="0" />
   </MTSInstance>
   <MTSInstance Name="invisible_larger_cube" Opacity="0">
      <Transform>
         <Scale x="1.01" y="1.01" z="1.01"/>
</Transform>

      <MTSGeometry Name="MTSSimple_Invisible" Type="MTSCube"/>

      <MTSHandle Event="MouseDrag" Action="MTSDrag"
Target="invisible_larger_cube" />

      <MTSHandle Event="MouseDrag" Action="MTSJavaScript"
Target="dragged(invisible_larger_cube::loc_)" />

      <MTSHandle Event="MouseLeftClick" Action="MTSJavaScript"
Target="stopdrag()" />

      <MTSHandle Event="MouseLeftUp" Action="MTSJavaScript"
Target="stopdrag()" />

   </MTSInstance>
</MTSInstance>
```

In the XML, the instance CubeA0 (the visible cube) has no scale tag meaning that the scale is 1, 1, 1 (the default). For the invisible cube, the scale is 1.01, 1.01, 1.01, making it slightly larger. The MouseDrag event is handled twice: The first event handler moves the invisible cube itself (this prevents the camera from rotating); the second event handler calls the JavaScript function dragged( ) to inform it that the invisible cube has moved and provide the new position of the invisible cube as a parameter to the dragged function.

© 2001 Viewpoint Corporation. All rights reserved.

Changing Parameters of Scene Text From the HTML Page

There are many ways to let your users customize the contents of a scene. If you were selling custom engraving and wanted to show different styles in a VET Web application, you might allow customers to experiment with different text styles. In this example, users can change options for a scrolling text marquee created from widget (3D hot spot) text.



For this text marquee, users can select text options from the HTML page.

Live example: http://cole.viewpoint.com/~ddavies/marquee/index.html

Scene files: http://cole.viewpoint.com/~ddavies/marquee/index.zip

On the HTML page, forms are used to offer different values for the text properties, and onclick and onChange tags call JavaScript functions that set property calls with the new value. There are numerous form items that allow setting of each property on demand.

To make the scene more visually appealing, it also implements a scrolling marquee effect. To do this, the setTimeout function calls the scroll_marquee( ) function, which changes the text 10 times per second. The scroll marquee function displays 20 characters from a string that has 75 total characters. Each time scroll_marquee( ) is called, it increments the start position in the string using an increasing offset. Thus, the scrolling effect is created by changing the text of the widget 10 times a second with an offset of 1 position each time.

Here's the standard definition of the instance variable:

```
<script language="JavaScript">
<!--

//Declare the variable that later is a handle to the plugin
//in the constructor call.

var pluginObj;
```

© 2001 Viewpoint Corporation. All rights reserved.

Here are the functions that allow setting of the text properties on demand:

```
//The following functions are used to set
//individual properties of the widget text.

//This function sets the "bold" property of the
//widget to either true or false ('1' or '0'),
//depending on the parameter.

function changebold(newbold)
{
   pluginObj.SetProperty('my_widget_text','bold',newbold,'mts_bool');
   pluginObj.Render();
}

//This function sets the italic property of the
//widget to either true or false ('1' or '0'),
//depending on the parameter.

function changeitalic(newitalic)
{
   pluginObj.SetProperty('my_widget_text','ital',newitalic,'mts_bool');
   pluginObj.Render();
}

//This function sets the underline property of the
//widget to either true or false ('1' or '0'),
//depending on the parameter.

function changeunderline(newunderline)
{
   pluginObj.SetProperty('my_widget_text','undr',newunderline,'mts_bool');
   pluginObj.Render();
}

//This function sets the strikeout property of the
//widget to either true or false ('1' or '0'),
//depending on the parameter.

function changestrikeout(newstrikeout)
{
   pluginObj.SetProperty('my_widget_text','strk',newstrikeout,'mts_bool');
   pluginObj.Render();
}

//This function sets the size property of the
//widget to the font size selected, depending on the parameter.

function changesize(newsize)
{
   pluginObj.SetProperty('my_widget_text','tsze',newsize,'mts_real');
   pluginObj.Render();
}

//This function sets the font property of the
//widget to the selected font, depending on the parameter.

function changefont(newfont)
{
   pluginObj.SetProperty('my_widget_text','font',newfont,'mts_str');
   pluginObj.Render();
}
```

© 2001 Viewpoint Corporation. All rights reserved.

```
//This function sets the color property of the
//widget to the selected color, depending on the parameter.

function changecolor(newcolor)
{
   pluginObj.SetProperty('my_widget_text','tclr',newcolor,'mts_pnt3d');
   pluginObj.Render();
}

//This function sets the text property of the
//widget to the selected text in the parameter.

function changetext(newtext)
{
   pluginObj.SetProperty('my_widget_text','text',newtext,'mts_str');
   pluginObj.Render();
}
```

Here is the code that handles the scrolling marquee effect:

```
//This variable is used as an offset into the marquee
//string. A fixed length of 20 characters are displayed
//at all times. The offset variable determine which 20
//are to be displayed.

var off_set=0;

//The marquee string. Only 20 characters are
//displayed at any given time.

var marquee_string = "                    This is an example of a Marquee
string sent to a widget.                    ";

//This is the function that keeps the marquee text
//scrolling. It is called repeatedly (due to setTimeout calls)
//and displays 20 characters from the marquee string.

function scroll_marquee()
{
   //This call takes a substring of 20 characters out of the
   //marquee string and calls changetext to set the widget
   //text equal to those 20 characters.

   changetext(marquee_string.substring(off_set, off_set+20));

   //Increase the offset so that next time this function is
   //called it displays a different 20 characters. It is
   //increased by one which creates a scrolling effect.

   off_set++;

   //This checks whether the 20 characters just displayed
   //are the last 20 characters in the marquee string. If so,
   //then the offset is reset to go back to the beginning.
   //If there weren't spaces at the beginning and the end
   //of the marquee string, then when this happens there would
   //be a jump from the last 20 characters to the first
   //20 characters, which would not look nice. You could add
   //some clever code here to check for that and handle it,
   //but it's just as easy to add 20 spaces to the beginning
   //and end of the string.
```

© 2001 Viewpoint Corporation. All rights reserved.

```
   if (off_set > marquee_string.length-20)
   {
      off_set=0;
   }

   //Set the timeout so that the marquee function is called
   //again in 1/10th of a second. Increasing this number
   //slows down the scroll. Decreasing speeds it up.

   setTimeout("scroll_marquee()",100);
}
```

This inline code is not part of a function and, therefore, runs automatically when the page loads. It is responsible for making the first call to the scroll_marquee( ) function. After the first call, the scroll_marquee function ensures that it is re-called after 100 milliseconds (one tenth of a second).

```
//This setTimeout starts the scrolling in 2 seconds time.

   setTimeout("scroll_marquee()",2000);
//-->
</script>
```

Here is the scene constructor that creates an instance of the Viewpoint Media Player (VMP):

```
</head>

<body>
<script language="javascript">
<!--

   //Construct a plugin instance.

   pluginObj = new MTSPlugin("layer2d.mtz", 500, 350, "../bkey.txt", "none",
"GenieMinimumVersion=50333494; ComponentMinimumVersion=50333494");
//-->
</script>
```

This is the definition of the form that allows the user to set any of the available properties.

Note: Be sure that the text options you offer the user are available on any system. For instance, use standard system fonts, as shown in this example.

```
<form name="theform">

<input name="sel_bold" type=checkbox SIZE=5
onclick="changebold(document.theform.sel_bold.checked ? '1' : '0')"> Bold
 

<input name="sel_italic" type=checkbox SIZE=5
onclick="changeitalic(document.theform.sel_italic.checked ? '1' : '0')">
Italic  

<input name="sel_underline" type=checkbox SIZE=5
onclick="changeunderline(document.theform.sel_underline.checked ? '1' : '0')">
Underline  
```

© 2001 Viewpoint Corporation. All rights reserved.

```
<input name="sel_strikeout" type=checkbox SIZE=5
onclick="changestrikeout(document.theform.sel_strikeout.checked ? '1' : '0')">
Strikeout  
<br><br>

Font <select name="sel_font"
onChange="changefont(document.theform.sel_font.options[document.theform.sel_fo
nt.selectedIndex].value)" >
  <option selected value="Arial">Arial
  <option value="Courier New">Courier New
  <option value="Times New Roman">Times New Roman
</select>
 

Size <select name="sel_size"
onChange="changesize(document.theform.sel_size.options[document.theform.sel_si
ze.selectedIndex].value)" >
  <option value=16>16
  <option value=20>20
  <option value=24>24
  <option value=28>28
  <option selected value=32>32
  <option value=36>36
  <option value=40>40
  <option value=44>44
  <option value=48>48
  <option value=52>52
</select>
 

Color <select name="sel_color"
onChange="changecolor(document.theform.sel_color.options[document.theform.sel_
color.selectedIndex].value)" >
  <option selected value="0.0 0.0 0.0">Black
  <option value="1.0 1.0 1.0">White
  <option value="1.0 0.0 0.0">Red
  <option value="0.0 1.0 0.0">Green
  <option value="0.0 0.0 1.0">Blue
</select>
</form>
```

© 2001 Viewpoint Corporation. All rights reserved.

Chapter 6:
Using JavaScript With Scene Interactors

Sending Events to the Scene Using JavaScript

Scene interactors can be super-charged by using JavaScript to post events from the HTML page. You can give the users the option to click buttons or icons on the HTML page that change the state of a scene or model, and script different actions and events for each state.

Live example: http://cole.viewpoint.com/~ddavies/postevent/index.html

Scene files: http://cole.viewpoint.com/~ddavies/postevent/index.zip

This example sends events from the JavaScript in the .html file to the embedded VET scene using the PostEvent function. The PostEvent function in JavaScript is the same as the PostMessage action available within the .mtx file ("event" and "message" are equivalent terms).

Note: This example uses an old MTS3Interface that didn't include support for the PostEvent function, which is why it needs to construct the command manually and use the Execute function. If you're using an MTS3Interface with version 4.x.x.x or above, you can simply use this syntax:

```
a.PostEvent('eventname','0')  //
```

where 0 is the delay.

```
<script language="javascript">
<!--
var a;

function
openBrWindow(theURL,winName,features){window.open(theURL,winName,features)}

function postevent(name, delay)
{
   //This function is needed because at this time
   //MTS3Interface doesn't support PostEvent yet
   //so we need to construct the plugin command here.

   var cmdstr = "PostEvent (" + name + "," + delay + ")";
   a.Execute(cmdstr);
}
//-->
</script>

</head>
<body>
<!-- FIRST OBJECT/EMBED -->
<script language="javascript">
   var alt = "none"
   a = new MTSPlugin("state1.mtx", 600, 500, "../bkey.txt", alt,
"GenieMinimumVersion=50333494; ComponentMinimumVersion=50333494");
</script>
<form>
   <br>
   <input type=button value="Send Inflating Event"
   onclick="postevent('ChangeToInflatingEvent','0')">
```

© 2001 Viewpoint Corporation. All rights reserved.

```
    <input type=button value="Send Opacitating Event"
    onclick="postevent('ChangeToOpacitatingEvent','0')">
    <input type=button value="Send Coloring Event"
    onclick="postevent('ChangeToColoringEvent','0')">
</form>
```

In the .mtx file, there is an interactor that handles inflating events and reflects the change of state visually by showing a texture on a plane that shows the state is set to *Inflating*.

```
<MTSHandle Event="ChangeToInflatingEvent"  EndState="Inflating"
Action="HandleInflatingEvent"  />
```

To find out more about Viewpoint scene interactors, download *Viewpoint Scene Interactors Reference Guide* from the Viewpoint Developer Central Web site at http://developer.viewpoint.com/.

© 2001 Viewpoint Corporation. All rights reserved.

Chapter 7:
Making a JavaScript Call in a VET Scene

Creating a Scene Animation That Displays Text on the HTML Page

With JavaScript, you can show annotations in the HTML page while the VET scene animations run. This could be a good way to illustrate and describe a product's functionality.

In this example, a scene animation is synchronized with text that describes the animation and is displayed in a field on the HTML page. As a cube changes color, an animator sends the name of the color to a text field.

Live example: http://cole.viewpoint.com/~ddavies/action2/index.html

Scene files: http://cole.viewpoint.com/~ddavies/action2/index.zip

In the scene *action2*, as an animator cycles through color changes on a cube, the .mtx file makes regular JavaScript calls to the function show_color( ) in the .html file. That function accepts a parameter (the color) in the form of a string. This is the code in the .mtx file:

```
<MTSTimeElem Type="ActionAnimator">
   <MTSJavaScript Func="show_color('red')"/>    <!-- time 0 -->
   <MTSJavaScript Func="show_color('green')"/>  <!-- time 1 -->
   <MTSJavaScript Func="show_color('blue')"/>   <!-- time 2 -->
   <MTSJavaScript Func="show_color('grey')"/>   <!-- time 3 -->
</MTSTimeElem>
```

At different time values (shown in commented form), the show_color function is called with different color name strings. In JavaScript, the following code causes the color name string to be displayed in the form, where the show_color( ) function changes the value field to reflect the new color:

```
<script language="JavaScript">
<!--
function show_color(temp)
{
   document.theForm.showcolor.value = temp;
}
//-->
</script>

<form name=theForm>
   Color <input type=text name=showcolor size=5 value="black" >
</form>
```

To find out more about using ActionAnimator, shown in the first code sample above, download *Viewpoint Experience Technology Scene Interactors Guide* from the Viewpoint Developer Central Web site: http://developer.viewpoint.com/.

© 2001 Viewpoint Corporation. All rights reserved.

Chapter 8: Using Automatically Generated Data in a VET Scene

You can further customize VET Web applications by implementing custom data, such as the date and time from the user's system. The examples in this section show you how to customize a scene by incorporating external data.

Implementing a Real-Time Digital Clock

This VET Web application feeds the time from the user's system clock into the scene to create a real-time digital clock. Although this digital clock scene is very simple, you could apply this example to a realistic model of a clock, for example. In this example, the system time displays as the text of a widget that is set to always visible.

Live example: http://cole.viewpoint.com/~ddavies/widgettime/index.html

Scene files: http://cole.viewpoint.com/~ddavies/widgettime/index.zip

Following is the JavaScript for this Web application:

```
<script language="javascript">
<!--
var pluginObj;
//This function sets the text property of the
//widget to the selected text in the parameter.
function changetext(newtext)
{
   pluginObj.SetProperty('my_widget_text','text',newtext,'mts_str');
   pluginObj.Render();
}
//This function is called once per second and gets the time
//from the system clock. It changes the text value of the
//hot spot (widget)to reflect the new time.
function tick()
{
      //Get the date/time and extract hours, minutes, and seconds.
   var tyme = new Date();
   var hours = tyme.getHours();
   var mins = tyme.getMinutes();
   var secs = tyme.getSeconds();
   //now extract individual digits.
   var dig1 = hours % 10;
   var dig0 = (hours - dig1) / 10;
   var dig3 = mins % 10;
   var dig2 = (mins - dig3) / 10;
   var dig5 = secs % 10;
   var dig4 = (secs - dig5) / 10;

   changetext(" " + dig0 + dig1 + ":" + dig2 + dig3 + ":" + dig4 + dig5)

   //Now the time update has been done, sleep for 1 second.
   //Call this function again in one second (1000 milliseconds).
   setTimeout("tick()", 1000); //come back here in 1 second
}
//-->
</script></head><body>
<script language="javascript">
<!--   //Construct a plugin instance.
   pluginObj = new MTSPlugin("time.mtx", 200, 150, "../bkey.txt", "none",
"GenieMinimumVersion=50333494; ComponentMinimumVersion=50333494");
//-->
</script>
```

© 2001 Viewpoint Corporation. All rights reserved.

Here's the XML for this example:

```
<MTSInstance Name="Simple_0" DoShadow="0">
   <Transform>
      <Scale x="0.0005" y="0.0005" z="0.0005"/>
   </Transform>
   <MTSGeometry Name="MTSSimple_0" Type="MTSCube"/>
   <MTSInstance Name="Layer2D_0">
      <Transform>
         <Scale x="0.2" y="0.2" z="0.2"/>
      </Transform>
      <LayerData Name="my_widget_text" Text=" 00:00:00" TextColor="0 0 0"
 Font="Courier" TextSize="32" AlwaysVisible="1"/>
   </MTSInstance>
</MTSInstance>
<MTSTimeElem Type="Keyframe" Name="start_clock" On="1">
   <Time> 0 0.001 </Time>
   <MTSTimeElem Type="ActionAnimator">
      <Trigger Target="Dummy"/>
      <MTSJavaScript Func="setTimeout('tick()',100)"/>
   </MTSTimeElem>
</MTSTimeElem>
```

Setting Scene Text to Display the Current URL

Live example: http://cole.viewpoint.com/~ddavies/location/index.html

Scene files: http://cole.viewpoint.com/~ddavies/location/index.zip

Text strings can be obtained from many sources to display in a scene. This example uses the reflection methods of JavaScript to obtain the URL of the page being viewed. The URL is then used in a SetProperty call on the *text* property of a widget. The same HTML page can be put on different Web sites and correctly identifies which site it is being served from. There is no absolute URL string in the .html file.

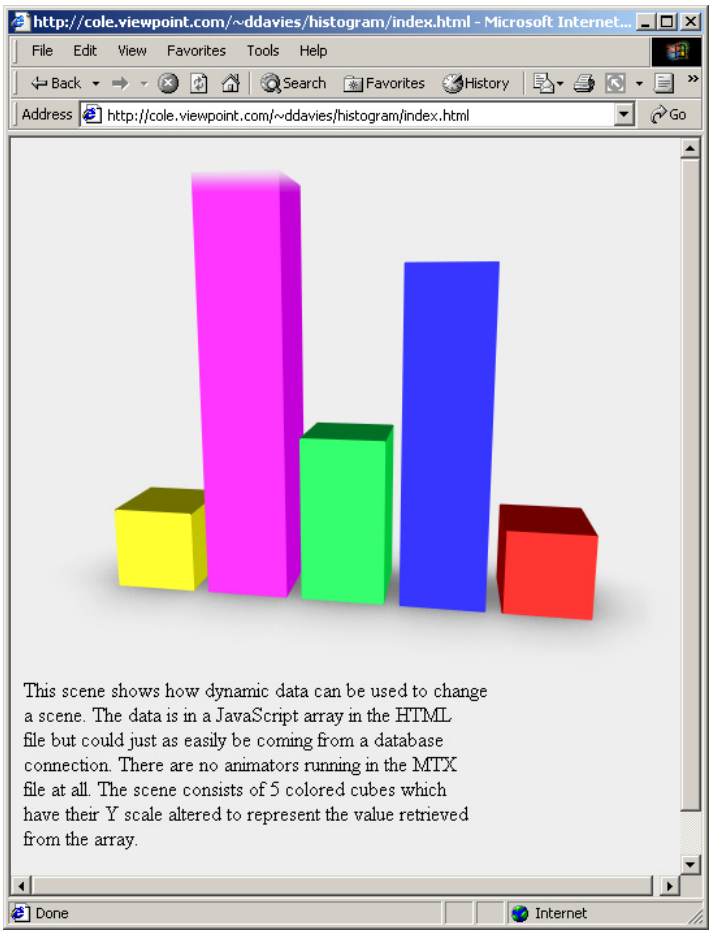
Here is the JavaScript code that sets the property of the widget:

```
<script language="JavaScript">
<!--

function changetext(newtext)
{
   pluginObj.SetProperty('my_widget_text','text',newtext,'mts_str');
   pluginObj.Render();
}

function showURL()
{
   var theURL = self.location.href;
   changetext(theURL);
}

//-->
</script>
```

© 2001 Viewpoint Corporation. All rights reserved.

Sending Dynamic Data to a Scene Via JavaScript

Give users an ultimate real-time experience by incorporating dynamic data in a scene. This example shows a simple histogram, but this technique could be used to create a chart comparing changing currency values, for instance.



Live example: http://cole.viewpoint.com/~ddavies/histogram/index.html

Scene files: http://cole.viewpoint.com/~ddavies/histogram/index.zip

To create the constantly changing histogram, JavaScript incorporates data to change elements in a scene—in this case, varying one dimension of primitive cubes. This example uses a simple array in the JavaScript to provide the simulated dynamic data, but data could come from a dynamic source.

The .mtx file in this example is very simple and defines 5 colored cubes spaced apart.

The JavaScript uses the data in the array to make SetProperty calls to the change the size of the cube in one axis only. It also uses a timer mechanism to change the size of the cubes once every second. This is achieved by using the standard JavaScript function setTimeout, taking a string and a timeout value as parameters. The string is a segment of JavaScript and can be a call to a function or simply the setting of a variable value. The timeout value is a number specifying the milliseconds to wait before executing the segment of JavaScript code in the string.

The final segment of JavaScript is the one that makes the first call to the update_chart( ) function. Without that call, the updating would never begin.

© 2001 Viewpoint Corporation. All rights reserved.

Here's the JavaScript for this example:

```
<script language="javascript">
<!--

//These following two JavaScript variables contain the data
//that is used to simulate data we could get from a
//database connection or some kind of live feed.

//The simulated_live_data variable contains 4 sets of 5 values.
//The 5 values are used to change the Y scale property of the
//five colored cubes. The first time the update_chart function
//is called, the sim_data_offset variable is equal to 0,
//so the first five values out of the simulated_live_data array
//are used. This means that the first cube's Y scale is
//set to 0.5, the second cube's Y value is set to 2.5
//the third cube's Y scale value is set to 1, the fourth
//cube's Y scale value is set to 2 and finally the fifth
//cube's Y scale value is set to 0.5. The Y scale values
//are set by a SetProperty call, which leaves the X and Z scale
//values unchanged at 0.5, it only changes the Y value.
//Once all cubes have a new Y value, the scene is rendered
//again to reflect the changes. Then, the offset variable, which
//is used to decide which set of 5 values to use from the
//simulated_live_data array, is incremented by 1 and checked
//to see if it should reset back to 0 because it has reached the
//end of the array.
//At the end of the update_chart value, a setTimeout call is
//made so that after 1 second the function is called
//again.

var simulated_live_data = new Array(0.5,2.5,1,2,0.5,  1,2,0.5,2.5,1,
1.5,1.5,1.5,2,1.5, 1,2,1.5,1.5,1);
var sim_data_offset = 0;

function update_chart()
{
   var scale = '0.5 ' + simulated_live_data[0+(sim_data_offset*5)] + ' 0.5';
   pluginObj.SetProperty('MTSInstance.Simple_0','scl_',scale,'mts_pnt3d');

   scale = '0.5 ' + simulated_live_data[1+(sim_data_offset*5)] + ' 0.5';
   pluginObj.SetProperty('MTSInstance.Simple_1','scl_',scale,'mts_pnt3d');

   scale = '0.5 ' + simulated_live_data[2+(sim_data_offset*5)] + ' 0.5';
   pluginObj.SetProperty('MTSInstance.Simple_2','scl_',scale,'mts_pnt3d');

   scale = '0.5 ' + simulated_live_data[3+(sim_data_offset*5)] + ' 0.5';
   pluginObj.SetProperty('MTSInstance.Simple_3','scl_',scale,'mts_pnt3d');

   scale = '0.5 ' + simulated_live_data[4+(sim_data_offset*5)] + ' 0.5';
   pluginObj.SetProperty('MTSInstance.Simple_4','scl_',scale,'mts_pnt3d');

   pluginObj.Render();
   if (++sim_data_offset >= 4) sim_data_offset = 0;
   setTimeout("update_chart()",1000); //call update_chart again in 1 second.
}

//declare the variable that will be a handle to the plugin
//in the constructor call later.

var pluginObj;
```

© 2001 Viewpoint Corporation. All rights reserved.

```
//-->
</script>
</head>
<body bgcolor="#eeeeee">
<script language="javascript">
<!--
//Construct a plugin instance.

   pluginObj = new MTSPlugin("chart.mtx", 500, 400, "../bkey.txt", "none",
"GenieMinimumVersion=50333494; ComponentMinimumVersion=50333494");

//make the first call to the update_chart function here
//to start things rolling.

   update_chart();

//-->
</script>
```

© 2001 Viewpoint Corporation. All rights reserved.

Chapter 9: Help, Resources, and Feedback

Viewpoint Developer Central: A Complete Resource

The Viewpoint Developer Central Web site is a complete resource for Viewpoint Experience Technology (VET) developers. At Developer Central you can get Viewpoint applications, support, production tips and techniques, tutorials, and user guides—to name just a few of the offerings there.

Go to Viewpoint Developer Central to

- Get Assistance For questions about using Viewpoint Experience Technology, click **Support** and go to the Viewpoint Forums.
- Get Examples Click **Create Content** and go to the Example Files.
- Subscribe to the Viewpoint Developer Newsletter Learn new production tips and techniques for creating 3D and rich media content for the Web with VET.
- Give Feedback About Viewpoint Applications Viewpoint Corporation values your feedback. Direct your comments and suggestions to the Viewpoint Forums.

You can also visit Viewpoint Corporation's main Web page for company news, links to Web sites featuring VET, and more.

Download Viewpoint Applications, Guides, and Tutorials

Viewpoint Developer Central is updated on an ongoing basis with the latest versions of its applications, user guides, and tutorials. Click **Create Content** to find links to the following.

Viewpoint Applications

You can download Viewpoint applications free of charge. Among applications available for download are

- Viewpoint Media Player (VMP) The Web browser plug-in necessary to view VET content with Netscape Navigator or Internet Explorer.
- Viewpoint Scene Builder An application designed to assemble and edit the content of a VET scene before its output to Viewpoint Media Files (.mts and .mtx/.mtz).
- Viewpoint Media Publisher (formerly called MTX2HTML) A utility that quickly creates an HTML page from a VET XML (.mtx/.mtz) file. This application provides a fast and convenient way for content creators to visualize a VET scene and animations within a Web page using the Viewpoint Media Player.
- Viewpoint Stream Tuning Studio An application designed to aid in the reducing file size of VET 3D content.
- Viewpoint Control Panel A utility designed to aid in the content creation, technical support, and development of VET-enabled Web sites and software.

User Guides and Tutorials

Get the most out of VET by learning with user guides and tutorials downloaded from Viewpoint Developer Central Web site.

Available are user guides covering the family of Viewpoint applications, including Scene Builder, Stream Tuning Studio, Control Panel, and Media Publisher (formerly, MTX 2 HTML). Also available for download is the *Viewpoint Experience Technology XML Authoring Overview* and *Viewpoint Experience Technology XML Reference Guide* documenting the XML that orchestrates all aspects of a VET scene.

Tutorials use sample files to lead you step-by-step through a specific aspect of creating with VET. Download tutorials, such as "Texture Animation," "Camera Animation," and "JavaScript: Animations and Browser Control."

© 2001 Viewpoint Corporation. All rights reserved.

Glossary

3D	Three-dimensional. An object or volume that exists in the dimensions of width, height, and depth.
hot spot	An area in a scene made up of a procedural shape (usually invisible) and created in Scene Builder. Hot spots are generally used to define a 3D area that when interacted with displays a text annotation, texture, or Flash movie. For instance, when a user points to a hot spot, a text-based annotation may appear.
action	Something that happens in a VET Web application. This can be triggered by a user clicking or pointing to a part of the scene (for example, an object), by other logic coded into the scene (such as the start state of a transition).
animation	A motion or transition added to a media atom or a group of media atoms over time. Examples include an object moving around a scene, transitions from one color or texture to another, or an object becoming visible.
Broadcast Key	A unique alphanumeric string issued by Viewpoint Corporation to companies or individuals licensed to broadcast VET content. The string is stored in a text (.txt) file that is referenced by VET-enabled Web pages. VET content without a Broadcast Key displays with a watermark.
bump mapping	A method of displaying textures not as a smooth surface, but as a rough surface that responds to different angles of illumination.
camera	The view from which a scene is rendered.
element	The complete statement of an XML command contained between an opening and closing tag. Elements include attributes and values and may contain nested elements, also known as subelements.
geometry	Defines all polygons making up an object.
global	Describes properties added to an entire scene.
interactors	Elements that allow the user to alter or interact with the scene by clicking or pointing to certain areas within the scene. Interactors are defined by XML code in the .mtx file for a VET Web application.
lightmap	An image that determines how light interacts with and scatters on the surface of an object. Material properties such as diffusion, specularity, and reflection are captured in the lightmap. The lightmap in any Viewpoint scene is what the camera sees in any reflective materials of an object. Any spherical image can be used as a lightmap image.
map	To apply a 2D image onto the surface of an object.
materials	Surfaces added to the mesh to give it a finished appearance unlike wireframe rendering.
media atoms	Components of a Viewpoint scene: 3D objects, material properties, sound, object movies, animators, interactors, and the definition of the 3D environment (that is, panoramas or the maps of environmental lightmaps).
mesh	See *geometry*.
MTS	A binary resource file (with a filename extension of .mts) containing all geometry, materials, and texture information for a VET scene. MTS is an open-specification 3D file format developed by MetaCreations and Intel Architecture Labs.
MTX	A Viewpoint XML scene file (with a filename extension of .mtx) that contains the hierarchical relationships between objects and other elements in the scene. This file is the script for staging the scene elements and usually references an .mts file. Also see XML.
MTZ	The filename extension (.mtz) for the compressed form of an .mtx file and the preferred format for Web-enabled Viewpoint content. Complex animations in an .mtx file can make file size large. Compressing these large .mtx files enables fast downloading of Viewpoint scenes.

© 2001 Viewpoint Corporation. All rights reserved.

properties	Attributes of a media atom. Also see media atoms.
render	The transformation of 3D data into 2D frames for display on a computer screen.
rotation	Moving an object around a specific center and axis.
scene	The highest level of the VET hierarchy (MTSScene tag in XML). Scene contains all elements of the .mtx and .mts files.
SWF	The Macromedia Flash movie file extension. May be pronounced “swif.”
texture	A picture on the surface, usually a JPEG or similar image file. This image file is rendered over polygons to give the object a realistic-looking surface.
tiling	The method of repeating a texture more than once across an object or part of an object. A tiled texture looks best if its edges seamlessly match up with each other, top to bottom and side to side. Tiling is a common method of using the smallest texture possible to cover a large area, such as a texture of a brick tiling across a large polygon or object to create an entire brick wall.
transforms	Transforms are position, rotation, and scale.
translate	To move the object along the x, y, or z axis in the scene.
VET	Viewpoint Experience Technology. Viewpoint Corporation’s unique technology that streams 3D and rich media content (media atoms) over the Internet via Viewpoint Media Player, a Web browser plug-in.
Viewpoint Media Player	The Web browser plug-in necessary to view VET content with Netscape Navigator or Internet Explorer.
VMP	See Viewpoint Media Player.
widget	See *hot spot*.
XML	Extensible Markup Language. A markup language for documents containing structured information with instructions for content (words, pictures, and so on) and the role that content plays (for example, content in a section heading has a different meaning from content in a footnote, figure caption, or database table). Viewpoint Experience Technology uses XML to define all properties of a scene.

© 2001 Viewpoint Corporation. All rights reserved.