Viewpoint

Viewpoint Media Player Release Notes

Version 3.0.15
May 2004

Viewpoint Media Player Release Notes

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: Caleb Hill, Nicolas Brun, Jamy Kokinda

Contributors: Michael Tuminello, Costa Touma, Ben Guihare, Jon Clegg, Daimen Duncan, Ed Peters, Barry Paul, Andrew Cook

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

Contents

Chapter 1: Introduction

This document contains information regarding the Viewpoint Media Player 3.0.15, 3.0.14, and 3.0.13 releases for the Windows operating system. This document is organized as follows:

- Chapter 1: “Introduction” — Overview of the document, changed behaviors and known issues in this release, and minimum system requirements.
- Chapter 2: “Release 3.0.15 New and Enhanced Features” — Introduces the new component features and XML tags, properties, and functions supported in Viewpoint Media Player 3.0.15 supports.
- Chapter 3: “Release 3.0.14 New and Enhanced Features” — Introduces the new component features and XML tags, properties, and functions supported in Viewpoint Media Player 3.0.14.
- Chapter 4: “Release 3.0.13 New and Enhanced Features” — Introduces the new component features and XML tags, properties, and functions supported in Viewpoint Media Player 3.0.13.
- Appendix A: “Help, Resources, and Feedback” — Describes Viewpoint’s free applications and documentation, and how to get technical support.
- Appendix B: “Viewpoint Content Creation Updates” — Describes new content creation techniques.

Changed Behaviors in this Release

This section describes changes from previous Viewpoint Media Player releases to the Viewpoint Media Player 3.0.13, 3.0.14, and 3.0.15 releases. Topics include:

- “Triggering an Animation”
- “MTSSetProperty/MTSAssignProperty/MTSCopyProperty”
- “Keyframe Animator”
- “Changing a Property on Assets during Player Execution”
- “Degrees Replacing Radians”
- “Crash Detection”
- “MTSImageStream Animations”
- “TalkNow Broadcast Key Requirements”
- “MTSLensFlares Instance Targeting”
- “MTSInteractor Hierarchy Relationships”
- “The MTSPreload Priority Tag”
- “The MTSRequireVersions Tag”
- “Flash”

> **Note:** Most Viewpoint Media Player 3.0.14 and 3.0.15 functionality is dependent on setting `<MTSScene Version="314">` or higher and installing the Viewpoint Media Player 3.0.15 component.

Triggering an Animation

Trigger functionality behaves differently based on the specified scene version.

Triggering an animation prepares the animation to run and turns it on. Two main changes in the triggering process from previous releases include:

- Loading values into a wildcard
- Handling the children on/off state

Loading Values into a Wild Card

With `<MTSScene Version="314">` or higher, using the `trgr` property or the `MTSTrigger` tag with an animation will load values into a wild card differently than when `<MTSScene Version="312">` or less.

Handling the Children On/Off State

With `<MTSScene Version="308">`, if the child animators are set to OFF and you start the parent, all the children will get set to ON and therefore get started as well. With `<MTSScene Version="312">` or higher, if you want all child animators to start when the parent is started, the child animators need to be ON.

Triggering an Animation Based on Scene Version Values

With `<MTSScene Version="314">` or higher, triggering an animation:

- updates the wildcards on the animator and all the children (and subchildren).
- rewinds the animator (not the children).
- turns on the animator (not the children).

With `<MTSScene Version="312">`, triggering an animation:

- updates the wildcards for the animator (not the children).
- rewinds the animator (not the children).
- turns on the animator (not the children).

And with `<MTSScene Version="308">`, triggering an animation:

- updates the wildcards for the animator and all the children (and subchildren).
- rewinds the animator and all the children (and subchildren).
- turns on the animator and all the children (and subchildren).

> **Tip:** You can recreate the `<MTSScene Version="308">` behavior (even with `<MTSScene Version="315">`) by setting the property, `trgr`, and specifying `Propagate="1"`. For more information on the `Propagate` attribute, see "The Propagate Tag and Property".

MTSSetProperty / MTSAssignProperty / MTSCopyProperty

The `MTSSetProperty`, `MTSAssignProperty`, and `MTSCopyProperty` tags behave differently based on the specified scene version.

With `<MTSScene Version="312">` or less, if one of these actions (`MTSSetProperty`, `MTSAssignProperty`, and `MTSCopyProperty`) fail, any remaining actions of the same `MTSAction` script still execute — this is inconsistent with other behaviors when an action fails.

In the following example, '`myobject`' is NOT an object in the scene. Therefore, `myobject.object` has no meaning — it is just a string. With `<MTSScene Version="314">` or higher, the `MTSSetProperty` action fails and all remaining actions do not execute.

```
<MTSSetProperty Target="myobject.object" Value="3" />
```

> **Tip:** If you set `IgnoreActionError="1"` when calling an action, you specify that even if an action fails, it should continue executing the remaining actions. For example:
>
> ```
> <MTSSetProperty Target="myobject.object" Value="3"
> IgnoreActionError="1" />
> ```
> For more information on `IgnoreActionError`, see "The IgnoreActionError Tag and Property".

MTSCopyProperty

With the introduction of new DOM functionality in release 3.0.14, `MTSCopyProperty` has been deprecated because now `MTSAssignProperty` can execute everything `MTSCopyProperty` can execute. For more information on this, see "Automatic Path Resolution".

Keyframe Animator

Keyframe animator functionality behaves differently based on the specified scene version.

With `<MTSScene Version="313">` or less, animations containing only one (1) keyframe and set to `<Time> 0 </Time>` do not work. To fix these animations, you either have to add another keyframe or change the time to something other than "0," as demonstrated below:

```
<MTSTimeElem Type="Keyframe" Name="animation_reset" On="0"
AutoStop="1">
    <Time> 0.001 </Time>
    <MTSTimeElem Type="ActionAnimator" >
        <animation_reset_action1/>
    </MTSTimeElem>
</MTSTimeElem>
```

With `<MTSScene Version="314">` or higher, animations work when there is only one key frame. This applies to both the `ActionAnimator` and `Timeline` examples shown below.

```
<!-- EXAMPLE 1: ActionAnimator -->
<MTSTimeElem Type="Keyframe"  >
    <Time> 0 </Time>
    <MTSTimeElem Type="ActionAnimator" >
        <animation_reset_action1/>
    </MTSTimeElem>
</MTSTimeElem>

<!-- EXAMPLE 2: Timeline -->
```

```
<MTSTimeElem Type="Keyframe"  >
    <Target Name="toto" Property="loc_" Timeline="T1" />
    <Time> 0 </Time>
    <Timeline Name="T1" Type="3D"> [0 0 0] </Timeline>
</MTSTimeElem>
```

Also, with `<MTSScene Version="314">` or higher, in cases where there are several keyframes, the first one is correctly executed. In previous releases, it was ignored and the process started evaluating at the second keyframe.

```
<!-- EXAMPLE 1: ActionAnimator -->
<!-- Do not need to setup a time 0 & 0.001 anymore -->
<MTSTimeElem Type="Keyframe"  >
    <Time> 0 1 </Time>

    <MTSTimeElem Type="ActionAnimator" >
        <!-- do not need MTSNoop anymore -->
        <animation_reset_action1/>
        <animation_reset_action2/>
    </MTSTimeElem>
</MTSTimeElem>

<!-- EXAMPLE 2: Timeline -->
<MTSTimeElem Type="Keyframe"  >
    <Target Name="toto" Property="loc_" Timeline="T1" />
    <!-- do not need to setup a time 0 & 0.001 anymore -->
    <Time> 0 1 </Time>
    <Timeline Name="T1" Type="3D"> [0 0 0] [1 0 0] </Timeline>
</MTSTimeElem>
```

> **Note:** In the examples above, the notes `<!-- do not need to setup a time 0 & 0.001 anymore -->`, refer to previous uses of a workaround for this functionality.

Changing a Property on Assets during Player Execution

With Viewpoint Media Player 3.0.14, changing a property on a material, texture, or geometry automatically re-renders the scene, even when using an action. You no longer have to set the property, `nerd`, on the scene to force the rendering of the scene.

Degrees Replacing Radians

When accessing a quaternion property using Euler values, the value units must be assigned in degrees or radians based on the .mtx file's scene version.

With `<MTSScene Version="313">` or lower, values must be provided in radians when setting Euler values (x, y, and z rotations) to a quaternion property.

With `<MTSScene Version="314">` or more, values must be provided in degrees instead of radians.

Crash Detection

Viewpoint Media Player release 3.0.14 contains a crash detection device that will detect an upcoming crash and, through the error handling process, stop Viewpoint Media Player but keep the web browser actively open. Upon detecting a crash, a log (VETlog.txt) is written to the root of the local drive (C:\VETlog.txt). It is recommended that this file be sent to **QA@viewpoint.com** for technical review.

MTSImageStream Animations

With `<MTSScene Version="312">` or lower, defining an `MTSImageStream` animator with an invalid target did not stop Viewpoint Media Player from streaming the file and did not stop the `MTSLoadDone` event from firing when the streaming completed.

With `<MTSScene Version="313">` or higher, if the target of an `MTSImageStream` animator is invalid, Viewpoint Media Player does not stream the file and the event is not triggered.

TalkNow Broadcast Key Requirements

Effective Viewpoint Media Player release 3.0.14 and higher, TalkNow content requires a Broadcast Key to play correctly. For more information on obtaining this broadcast key, contact sales@viewpoint.com.

MTSLensFlares Instance Targeting

With `<MTSScene Version="314">` and higher, the `MTSLensflares` feature can target an instance that is in a hierarchy in addition to an instance that is a direct child of the root instance.

MTSInteractor Hierarchy Relationships

With Viewpoint Media Player release version 3.0.13, interactors can be parented. This means that now you can create a hierarchy of interactors. As a result, if a parent interactor is inactive, its children automatically are inactive, too.

To activate or deactivate an interactor, use the tag `Active` or the property `actv` as follows:

```
<!--Here, 'myChild1' is inactive by default. The first time the event
'MouseLeftDown' is fired, 'myParent' interactor catches it and turns
on 'myChild1'. On the next 'MouseLeftDown' event, both interactors
catch the event. -->

<MTSInteractor Name="myParent" NeverHandle="1" />
    <MTSHandle Event="MouseLeftDown" Action="MTSAssignProperty"
     Target="MTSInteractor.myChild1::actv" Value="1" />

    <MTSInteractor Name="myChild1" Active="0" />
        <MTSHandle Event="MouseLeftDown" Action="myAction2" />
    </MTSInteractor>

    <MTSInteractor Name="myChild2" />
        <MTSHandle Event="myEvent1" Action="myAction3" />
        <MTSInteractor Name="mySubChild" />
            <MTSHandle Event="myEvent2" Action="myAction4" />
        </MTSInteractor>
    </MTSInteractor>
</MTSInteractor>
```

The MTSPreload Priority Tag

With Viewpoint Media Player 3.0.15, the order in which preloaders load can be specified.

The MTSRequireVersions Tag

With Viewpoint Media Player 3.0.15, the minimum version requirements for a scene's components can be specified in XML code, ensuring that the player's components and classes meet the specified requirements.

Flash

Viewpoint's Flash functionality has been enhanced significantly from version 3.0.12 to 3.0.14, including the following behavior changes:

- "Movie Clip Timing"
- "Buttons"
- "ActionScript"
- "Nesting Tinting"
- "Masking"
- "MouseDrag Event"

Note: If you encounter Flash-specific behavior issues not mentioned in these release notes, please send your feedback and content files to flashqa@viewpoint.com for technical review.

Movie Clip Timing

- Movie clip timing now behaves as in the Macromedia Flash player. As a result, the behavior of content with movie clips in the 3.0.12 release and this release may differ. Specifically, in 3.0.12, when a movie loops, clips inside the movie are reset to their starting frame. In 3.0.14, the time line for clips is fully independent of the main time line, as is the case with the Macromedia Flash player. So when a movie loops, subclips continue to play. In 3.0.12, if a 10 frame movie contained a movieclip with 20 frames, you would never see the last 10 frames of the clip.
- Timing in general is greatly improved. Movie clips always start on frame 1 (which was unreliable in 3.0.12), and multiple movie clips all enter a new frame at the same time (in 3.0.12 multiple movie clips became desynchronized over time).
- In 3.0.12, the last frame of a.swf file does not render. In 3.0.14, the last frame renders correctly.

Buttons

- In the 3.0.12 release, the "hit" frame was ignored, and the hit area was the union of the area of the "down", "up", and "over" frames. In the 3.0.14 release, the "hit" frame is used to determine the clickable area of the button, as in the Macromedia Flash player.
- In the 3.0.12 release, buttons with an alpha setting of 100% (fully transparent) did not work as buttons. This has been fixed in release 3.0.14.
- In the 3.0.14 release, reliability of button functionality and ActionScript execution from buttons is greatly improved. In the 3.0.12 release, buttons only worked on the main time line, and functionality was limited. In the 3.0.14 release, buttons work inside of clips and additional functionality, including animated button states, is supported.

ActionScript

- In the 3.0.12 release, Viewpoint Media Player supported only a subset of ActionScript. ActionScript support is significantly more complete in release 3.0.14. In general, expect behavior to be similar to Macromedia Flash, provided you avoid syntax unsupported by Viewpoint Media Player. Support for operators, objects, methods, and properties is greatly expanded, as detailed in "Macromedia® Flash™ Support".

- In the 3.0.12 release, regarding the Macromedia Flash `fscommand()` and naming, posting messages with the same names as ActionScript commands (such as “play” and “stop”) worked sporadically. This issue has been resolved in release 3.0.14.
- In the 3.0.12 release, the `fscommand()` call was supported when Flash played only in the background (preanimator) and foreground (postanimator) layers of a scene. In the 3.0.14 release, `fscommand()` works in these two layers AND when Flash is applied as a texture.

Nesting Tinting

Nested tinting did not work in the 3.0.12 release. In release 3.0.14, it is fully functional as in the Macromedia Flash player. For example, a clip with 100% blue tint inside a clip with 100% black tint will be black in release 3.0.14 and in the Macromedia Flash player.

Masking

In the 3.0.12 release, masks inside a movie clip did not work. In release 3.0.14, masks are supported in most contexts. For specific details, see “Known Issues in this Release”.

MouseDrag Event

Prior to the 3.0.14 release, the `MouseDrag` event did not work with Flash animations. The workaround was to set up the `MouseDrag` event to call an action that changes the `MouseDrag` event to `MouseMove` prior to triggering the Flash file. For example:

```
<!--Here, we set the pre-action, "Myobject," to the real action,
"Flashanim" -->
<MTSAction Name="Myobject">
    <MTSAssignProperty Target="MTSEvent::name" Value="MouseMove" />
    <MTSAction Name="Flashanim" />
<MTSAction />

<!--Here, the event, MouseDrag, is specified to call the "Myobject"
action, which effectively changes it to MouseMove before triggering
the "Flashanim" action. The other listed events directly trigger the
"Flashanim" action. -->

<MTSInteractor>
    <MTSHandle Event="MouseDrag" Action="Myobject" />
    <MTSHandle Event="MouseLeftDown" Action="Flashanim" />
    <MTSHandle Event="MouseLeftUp" Action="Flashanim" />
<MTSInteractor />
```

The `MouseDrag` event is fixed in release 3.0.14; the workaround no longer is needed. For example:

```
<!--Here, the MouseDrag event can directly call "Flashanim" without
first calling the pre-action, "Myobject" to change its name. -->

<MTSInteractor>
    <MTSHandle Event="MouseDrag" Action="Flashanim" />
    <MTSHandle Event="MouseLeftDown" Action="Flashanim" />
    <MTSHandle Event="MouseLeftUp" Action="Flashanim" />
<MTSInteractor />
```

Known Issues in this Release

Following is a descriptive list of known issues for the Viewpoint Media Player 3.0.14 release.

Propagating an Action Using the DOM

With release 3.0.14, a new action-specific tag and property, `Propagate`, is introduced along with new DOM functionality. Both of these features are explained in detail in "SceneComponent Features".

In release 3.0.14, you should be able to use the new DOM syntax in correlation to the `Propagate` tag, demonstrated here:

```
<!--Note the Target value, "Simple_0.opac" uses the new DOM syntax (.).
-->

<MTSInteractor Name="setint">
    <MTSHandle Event="MouseLeftDown" Action="MTSSetProperty"
     Target="Simple_0.opac" Value="0.5" Propagate="1"/>
</MTSInteractor>
```

However, it is known that this syntax does not work correctly. Instead, use the pre-3.0.14 syntax as follows:

```
<!--Note the Target value, "Simple_0::opac" uses the previous DOM
syntax (::). -->

<MTSInteractor Name="setint">
    <MTSHandle Event="MouseLeftDown" Action="MTSSetProperty"
     Target="Simple_0::opac" Value="0.5" Propagate="1"/>
</MTSInteractor>
```

Integrating Vector Graphics with Zoom

- To integrate Macromedia Flash/SVG with ZoomView, you must keep the aspect ratios the same and the Flash/SVG size must be 1/20th of the ZoomView size (i.e., ZoomView Height/20 and ZoomView Width/20).

  For example, if the ZoomView image is 25000 x 10000, the Flash/SVG must be 1250 x 500.

Flash Masking/Tinting

- A movie clip (or graphic) with a mask, in turn, cannot be masked. Also, masked movie clips cannot be tinted (tinting applied to these clips is ignored).
- If the graphic (shape) used in a mask-layer has an alpha<100%, Viewpoint Media Player incorrectly calculates the mask.

  To resolve this issue, make sure your shapes have an alpha=100% for the mask layer.

Flash Scripting

When using `fscommand()` to pass a variable string, the following syntax fails:

```
var myEvent = "eventName";
fscommand ("PostMessage","Message="+myEvent);
```

The workaround for this issue is to send only one string in `fscommand()` as follows:

```
var myEvent = "eventName";
fscommand ("PostMessage Message="+myEvent);
```

Flash Movie Clip

- If a movie clip is initialized on a `frame >=2`, `onClipEvent(load)` is executed before the ActionScript of the frame of the time line containing the clip; this is opposite of how it should work.

  To rectify the order for `onclipEvent(Load)`, put the movie clip on stage on frame 1 of the time line, or make sure that the frame script you need to execute before the initialization of the clips is on an earlier frame.

- In an `onclipEvent(load)` event handler or in code on the first frame of a movie clip, access to properties of `"this"` returns incorrect values (`0` or `NaN`). This works correctly on frame 1 of the time line. As a workaround, do not try to get the properties of a clip in the same frame as it first appears. If the clip appears on frame 3, wait until frame 4 to access its properties.

- Some movie clip properties are incorrectly retained in Viewpoint Media Player. To resolve this issue, when a movie clip (clipA) is reloaded, re-initialize movie clip properties, or take other steps to make sure the clip is fully reinstantiated (like going to a frame before the clip's appearance).
  For example, with a movie clip (Clip A) which contains a movie clip (Clip B), where Clip B first appears on frame 4 of clip A. If you are on frame 10 of Clip A (Clip B is already instantiated and on stage), and use a `gotoAndPlay(4)` command, Clip B is not correctly reinstantiated and continues to have the properties it had in frame 10. Instead, go to frame 3 (before Clip B loads), or reset the properties of Clip B explicitly with ActionScript.

- In 3.0.14, ActionScript event execution order incorrectly relies on clip depth and other factors. As a general tip, avoid using `onClipEvent()` actions when possible in favor of functions on the main time line.
  For example, if you have 10 clips that need to be initialized with properties, use a function with a for loop to assign properties rather than 10 instances of `onClipEvent(load)`. If possible, structure your events in such a way that there is no question as to what the execution order is. To troubleshoot problems, have events output a message to a text field to be sure that the events are executing in the correct order.

Flash Dynamic Text

- Multiline/single line/word wrap parameter is ignored. It is always considered as multiline + wordwrap.

- Dynamic text fields cannot be set to the property of a clip. For example, `clip._x`. Instead, set a variable equal to the property and then set the text field equal to the variable.

- Scrolling functions `scroll()` and `maxscroll()` are unsupported. Moving a movie clip containing text behind a mask would be one way to implement text scrolling functionality within the bounds of what is allowed by 3.0.14.

- Text is always antialiased, even system text as accessed by setting your font choice to `"_sans"` or `"_serif"`. Also, pixel fonts are antialiased. The only workaround is to substitute graphics for text.

- Editable text is not supported.

- Text spacing and kerning are different, and there are some rendering differences with text between 3.0.14 and the Macromedia Flash player.

Flash LoadMovie/LoadVariables

- All functions related to the loading of external movies or variables (`loadMovie()`, `loadMovieNum()`, `LoadVariables()`, etc.) are not functional. Variables can be passed from Viewpoint Media Player via JavaScript.
- 'Delete this result' is not reliable
- The scope followed by the delete in Viewpoint Media Player is different from in the Macromedia Flash Player. Viewpoint Media Player can delete local variables. in Macromedia Flash, ‘delete’ never deletes local variables.

  For example:

  ```
  a = 50; function test(){var a = 60;delete a;} test();trace(a);
  ```

  - Macromedia Flash, `trace(a)` returns Undefined because `delete a` deleted global variable.
  - In Viewpoint Media Player, `trace(a)` returns 50 because `delete a` deleted the local variable instead of the global.

Cursor Issues

- The hand cursor that normally displays when rolling over a clickable area, such as a button, does not display in Viewpoint Media Player unless a message is explicitly passed to Viewpoint Media Player to change the cursor.
- `Mouse.hide()` hides the cursor for the entire browser; be sure to use it in conjunction with `Mouse.show()` to make it visible when it leaves the affected area.

Performance Issues

- When animating a limited number of clips, Viewpoint Media Player performance is comparable to the Macromedia Flash player. In release 3.0.14, using ActionScript to animate a large number of clips degrades performance.

Color/Rendering issues

- In release 3.0.14, the background of a .swf file is transparent by default. Use the property `FillBackground=1` to show the background, or manually draw a rectangle of the right color on the bottom layer of the .swf file.
  For more information on `FillBackground`, see “The FillBackground Property”.
- While the `Color` object is mostly reliable, there are occasional differences in rendering when using it.

Miscellaneous Issues

- Not all printing functions are supported.
- Smart Clips are not supported.
- `toggleHighQuality()` is not supported.
- `escape()` and `unescape()` are unsupported. You can use JavaScript for this functionality, if required.

- In Flash 4 it was possible, although not necessarily advisable, to name clips with a period in the title (for example, `myPoorlyNamed.clip`). While the Macromedia Flash Player supports this naming convention for backwards compatibility, Viewpoint Media Player release 3.0.14 does not (use `myValidlyNamedclip`, for example).
- There is no support of OOP functionality in ActionScript in release 3.0.14.
- There is no support of XML, XMLSocket, XMLNode, Sound, or Selection objects in release 3.0.14.
- The `Date` object is partially supported, as detailed in "Macromedia® Flash™ Support".
- `Key.isToggled()` does not work correctly — it functions like `Key.isDown()`.
- In rare instances (for example, `Math.atan2(-Infinity, Infinity)`) methods of the `Math` object may return different values than the Macromedia Flash Player.
- `getVersion()` returns the following inside Viewpoint Media Player:
  - Platform and Flash version (currently 5,0,0,0)
  - VET version
  - XML version

  For example:
  - WIN 5,0,0,0
  - VET 3,0,14,141
  - XML 311
- In some instances, like `targetPath()`, Viewpoint Media Player 3.0.14 or higher returns `"_root"` in place of `"_level0"`.

Minimum System Requirements for this Release

- Microsoft® Windows® 98, Windows 2000, Windows Millennium Edition, Windows NT® 4.x, or Windows XP
- Netscape Navigator® 7.0
- Microsoft® Internet Explorer 5.x
- AOL® 7.0
- 64 MB RAM
- Pentium® II 300 MHz processor or faster

Chapter 2: Release 3.0.15 New and Enhanced Features

Viewpoint Media Player functions as a graphics operating system, a central hub comprised of several components (.dll files) that enable disparate media types, such as Flash, audio streaming, and video, to seamlessly integrate into one scene. Due to its component design, Viewpoint Media Player downloads only those components that are required to view the content accessed by a user, thereby reducing download time and interruptions.

This chapter summarizes the new and enhanced features introduced in Viewpoint Media Player release 3.0.15 according to the following component categories, including brief descriptions of all newly released tags, properties, and functions.

- "SceneComponent Features" — The SceneComponent component now supports a new action feature enabling conditional logic to execute actions in MTX code based on variable values. It also supports a new `MTSBaseComponent` function for clearing objects and their children from a scene.
- "SceneCapture Features" — This new component enables Viewpoint Media Player to capture a scene and save it to a local or remote host. It also introduces new render functionalities.
  The SceneCapture component is declared in MTX code as follows:
  ```
  <MTSTimeElem Type="VMPExtremeShot" Name="MyImage" />
  ```

SceneComponent Features

SceneComponent is the main component of Viewpoint Media Player. Among other functions, this component parses MTX files, creates objects in scenes, evaluates animations, manages events, and prepares scenes to be rendered by the renderer.

Viewpoint Media Player 3.0.15 introduces a new action tag and property to SceneComponent: `MTSCompareProperty`. It also introduces a new `MTSBaseComponent` function for clearing objects and their dependencies from a scene.

The MTSCompareProperty Action

The `MTSCompareProperty` action tests whether two specified values are the same. This new feature allows you to execute conditional “if” logic in a scene to decide the actions to execute based on variable values...

Tag and Property	Description
`<MTSCompareProperty>`	Tests two specified sets of values to find if they are equal. If the values are equal, all subsequent actions in the scene execute. If they are not equal, the `MTSCompareProperty` action fails and all subsequent actions in the scene do not execute.
`Prop`	Specifies the property whose values are to be compared.
`Value`	Specifies the values that are matched against those of the specified property. If the values match, the action succeeds.
`Equal`	`Equal=”0”` reverses the logic of the action so that the action succeeds if the value comparison does not yield a match.

The following sample MTX code demonstrates how to use this tag and property:

```
<--EXAMPLE 1: Here, we test whether the Translation of the MTSInstance
named “cub” is equal to 0 0 0 -->

<MTSCompareProperty Prop="MTSInstance.cub.Translation" Value="0 0 0"/>

<!--EXAMPLE 2: Here, we test whether or not the Translation of the
MTSInstance named “cub” is equal to the Translation of the MTSInstance
named “sphere.” -->

<MTSCompareProperty Prop="MTSInstance.cub.Translation"
Value="MTSInstance.sphere.Translation" />

<--EXAMPLE 3: It is possible to revert the logic of MTSCompareProperty
and make the action succeed if the two set of values are NOT equal,
setting Equal=”0”.

<MTSCompareProperty Prop="MTSInstance.cube.Translation" Value="0 0 0"
Equal="0" />
```

MTSBaseComponent Functions

Unlike properties, functions do not store a state of an object, but instead trigger an action on the object. Currently, very few objects feature these built-in functions. For more information on functions, see "Functions".

Most object functions relate to existing objects, such as `MTSBaseComponent`, and are called via the `<VETDispatchCall Function>` command.

Note: The `<VETDispatchCall Function>` tag can be researched at Viewpoint XML Reference Guide.

Viewpoint Media Player 3.0.15 introduces the following function.

RemoveObjectAndDependencies

This `MTSBaseComponent` function is an enhancement to the `MTSBaseComponent RemoveObject` function. For more information on this function, see "RemoveObject".

Function	Description
`RemoveObjectAndDependencies`	Removes a specified object and any dependent of this object (children, geometries, textures, and assets in general) from a scene and memory (RAM).

The following sample MTX code demonstrates how to use this function:

```
<!--EXAMPLE 1: Here, the RemoveObjectAndDependencies function removes
the specified object, 'toto.' -->

<VETDispatchCall
Function="MTSBaseComponent::RemoveObjectAndDependencies(toto)" />

<!--EXAMPLE 2: Here, the RemoveObjectAndDependencies function removes
every object, geometry, and texture from the scene. -->

<VETDispatchCall
Function="MTSBaseComponent::RemoveObjectAndDependencies(MTSRootInstanc
e)" />

<!--EXAMPLE 3: Here, the RemoveObjectAndDependencies function removes
every animator from the scene. -->

<VETDispatchCall
Function="MTSBaseComponent::RemoveObjectAndDependencies(MTSRootAnimato
r)" />
```

SceneCapture Features

The newly released Viewpoint SceneCapture component introduces Viewpoint Media Player's first image capture functionality, making it possible for you to create and save scene images in a fast and easy manner without the hassle of using print screen functions and image manipulation programs to extract images.

The Viewpoint SceneCapture component can help improve content authoring, verify how users interact with online advertisements, and render mirror-like visual effects in Viewpoint scenes.

Key Features

Viewpoint SceneCapture enables you to:

- Capture scene images at high-resolutions
- Crop and size captured scene images to your specifications
- Capture scene images from multiple camera angles
- Save captured scene images as .jpg files to remote and local locations
- Render mirror effects within a Viewpoint scene

Viewpoint SceneCapture enables you to create high-resolution .jpg files that can be cropped, sized, and saved according to your specifications while also allowing you to render planar or even curved mirror effects.

To utilize the Viewpoint SceneCapture component in content, familiarize yourself with the suite of Viewpoint XML functions and attributes that corresponds to "The VMPExtremeShot Animation Type". This animation type corresponds to SceneCapture functionality in MTX code.

The VMPExtremeShot Animation Type

Within your scene's control file (.mtx file), you can use the Viewpoint SceneCapture component via the Viewpoint animation type, `VMPExtremeShot`. Like all Viewpoint animators, `VMPExtremeShot` pertains to the `MTSTimeElem` element and is declared in MTX code in the same manner as all `MTSTimeElem` animator types:

```
<MTSTimeElem Type="VMPExtremeShot" Name="MyImage" />
```

Note: Viewpoint SceneCapture saves all captured scenes as .jpg files.

For example, in MTX code use the `VMPExtremeShot` animation type to capture a scene that might contain Flash, 3D content, and/or other media resource types and save the resulting .jpg file as specified in the MTX code.

Overview of VMPExtremeShot Functions and Attributes

The SceneCapture component includes a robust suite of functions and attributes that you declare separately in your scene's .mtx file(s). For more information on Viewpoint functions and XML syntax, see "Document Object Model (DOM) Syntax".

Compatibility: Though introduced in Viewpoint Media Player 3.0.15, Viewpoint SceneCapture is compatible with previous player versions, extending back to Viewpoint Media Player 3.0.8, with the exception of two SceneCapture attributes, `Mirror` and `Camera`, which require Viewpoint Media Player 3.0.15.

The VMPExtremeShot Functions

The scene capture and save functionality of the SceneCapture component are accessed via these `VMPExtremeShot` functions:

Function	Description
`VMPCapture`	Captures an image from a scene.
`VMPFlushCapture`	Flushes a captured image from the user's memory.
`VMPLocalPersist`	Saves captured data from a scene onto the user's local drive. When saving captured images locally, a **File Save** dialog box automatically displays, requiring the user to enter the name of the file to be saved locally. This cannot be disabled.
`VMPRemotePersist`	Saves and sends captured data from a scene onto a remote server. To enable this functionality, obtain a valid Broadcast Key. To obtain a Broadcast Key file for the Viewpoint SceneCapture component, see Viewpoint Developer Central.

Note: The property, `DISP`, can be used to invoke all `VMPExtremeShot` functions using Viewpoint Media Player version 3.0.8. For information on the syntax used to call these functions in MTX code, see "Declaring VMPExtremeShot Functions in MTX Code".

Declaring VMPExtremeShot Functions in MTX Code

The particular syntax you use to declare these functions in your .mtx file(s) can vary:

- **Recommended syntax** Valid for Viewpoint Media Player versions 3.0.11 and higher, the `VETDispatchCall` command calls the `VMPExtremeShot` functions. For example:

```
<MTSHandle Action="VETDispatchCall" Function="ShotIt::VMPCapture()"
  Event="MouseRightClick" />
```

Note: For more information on `VETDispatchCall` and Viewpoint functions in general, see the *Viewpoint XML Reference Guide* from Developer Central.

- **Alternative syntax** Valid for Viewpoint Media Player versions 3.0.8 and higher, the `MTSSetProperty` tag calls the `VMPExtremeShot` special functions via the `DISP` property. For example:

```
<MTSSetProperty Target="capture::DISP" Value="VMPCapture"
Event=MouseLeftClick" />
```

The VMPExtremeShot Attributes

The SceneCapture component enables you to specify how a scene image is captured or mirrored via these principal attributes of the `VMPExtremeShot` animation type:

Tag and Property	Description
`Path`	Specifies the path to a remote host where a captured data file is sent.
`LocalFileName`	Defines the default .jpg file name that displays in a dialog box for locally-saved images.
`AntiAliasingPassesCount`	Specifies how many progressive anti-alias passes the SceneCapture component makes on an image being captured.
`Quality`	Specifies the compression of the .jpg file that is created from captured content. The lower the value (range 0 to 100) you assign this attribute, the more you compress the .jpg file.
`Size`	Specifies the size of the image that is captured (or mirrored) from Viewpoint Media Player.
`Rect`	Crops the image that is created from captured (or mirrored) Viewpoint Media Player content.
`FillColor`	Fills a captured (or mirrored) image's background with a specified color.
`FillAlpha`	Specifies the transparency of a captured (or mirrored) image's background.

Tag and Property	Description
`Camera`	Specifies an alternative camera to the scene's default camera by which an image is captured (or mirrored). The alternative camera, `VETCamera`, for example, can produce image captures that are different to the camera angle used by the scene. To do this, your content should contain several cameras.
`Mirror`	Creates and displays a mirror image effect of a scene's content. This attribute defines the instance (flat plane surface) that is used as a mirror. Also, you can use a curved surface, but it does not reflect an accurate scene reflection.

Viewpoint SceneCapture Sample MTX Code

The following MTX code sample illustrates a simple scene using the Viewpoint SceneCapture component.

```
<!--Here, two functions are called: the first, VMPCapture, captures
the current scene when the user left mouse clicks. The second,
VMPLocalPersist, saves the scene capture to the user's local drive when
the user right mouse clicks. -->

<!--Here, we declare the VMPExtremeShot animator, “Shot” -->

<MTSTimeElem Type="VMPExtremeShot" Name="Shot" Quality="60"
AntiAliasingPassesCount="15" Rect="12 12 80 80" />

<MTSInteractor>
    <MTSHandle Action="VETDispatchCall" Function="Shot::VMPCapture()"
     Event="MouseLeftClick" />

    <MTSHandle Action="VETDispatchCall"
     Function="Shot::VMPLocalPersist()" Event="MouseRightClick" />
</MTSInteractor>
```

Chapter 3: Release 3.0.14 New and Enhanced Features

This chapter summarizes the new and enhanced features introduced in Viewpoint Media Player release 3.0.14 according to the following component categories, including brief descriptions of all newly released tags, properties, and functions.

- "SceneComponent Features" — The SceneComponent component now supports full DOM syntax and several new animation features among others.
- "Flash Features" — The Flash component now supports:
  - More Flash ActionScript functions, allowing more effective communication between Flash and Viewpoint Media Player.
    Flash is declared in MTX code via `<MTSTimeElem Type="SWFView">`.
  - More native SVG tags, and new Viewpoint-enhanced SVG tags.
    SVG is declared in MTX code via `<MTSTimeElem Type="SWFView">`.
- "Video Features" — The new Video component enables cost effective and reliable video deployment.
  Video is declared in MTX code as `<MTSTimeElem Type="iVideo">`.
- "TalkNow Features" — The TalkNow component now supports new volume controls.
  TalkNow is declared in MTX code as `<MTSTimeElem Type="VMPSpeech">`.

SceneComponent Features

SceneComponent is the main component of Viewpoint Media Player. Among other functions, this component parses MTX files, creates objects in scenes, evaluates animations, manages events, and prepares scenes to be rendered by the renderer.

Viewpoint Media Player 3.0.14 introduces several key new SceneComponent features and a many enhancements for existing features, including:

- "Document Object Model (DOM) Syntax" — The DOM syntax, already partially available since release 3.0.12, has been greatly enhanced to give increased authoring freedom to the content developer.
- "VETOrbitManipulator" — This new feature manipulates any object in a scene that has a matrix property type, such as VETCamera.
  Viewpoint Instance Manipulator is declared in MTX code as `<VETOrbitManipulator Name="CameraManipulator" >`.
- "VETSequencer" — This new feature synchronizes animations that do not run for the same length of time.
  `VETSequencer` is declared in MTX code as `<VETSequencer Name="coco" On="1" >`
- "VETStreamCase" — This new feature dynamically chooses the MTX child node to execute based on the type of media file streamed from a host server, even if the exact file type is not known.
- "MTSBaseComponent Functions" — This existing feature includes two new functions that clear objects and more from a scene.
- "VETRandomGenerator Tags and Properties" — This existing feature includes two new tags and properties for random string generation.
- "Viewpoint Media Player Events" — Includes new and enhanced events and introduces a new syntax using the DOM to collect events.
- "MTSTimeElem Tags and Properties" — Includes two new tags and properties, including one used for specifying targets and another for animator download information.
- "The Propagate Tag and Property" — This new feature extends the action commands, `MTSSetProperty` and `MTSAssignProperty`, to allow for property propagation throughout a scene hierarchy.
- "The IgnoreActionError Tag and Property" — This new feature enables you to continue executing actions even if one of them fails.
- "MTSInstance Tags and Properties" — Includes many new tags and properties for use on geometry and materials.
- "MTSMaterial Tags and Properties" — Includes new tags and properties that access texture information.
- "MTSCookie Tags and Properties" — Includes support for one new `MTSCookie` tag, `Extend`.

Document Object Model (DOM) Syntax

The DOM is a way to access properties (or attributes) of objects in a scene. The new DOM syntax now gives content developers increased authoring freedoms.

Pre-3.0.14 DOM Syntax

Since the very first version of Viewpoint Media Player, a property could be accessed in the following way:

```
<!--Here, we set the property loc_ (location, or position in space) of
an object named 'cube' in the scene. -->

<MTSSetProperty Target="cube::loc_" Value="1 2 3" />
```

The DOM syntax solves the issue for when an animator shares the same name as an instance in the scene. For example, to specify the type of the object to access, you could declare:

```
<!--In this example we specify that we want to access the property
named 'loc_' of the object named 'cube' which is an instance.

<MTSSetProperty Target="MTSInstance.cube::loc_" Value="1 2 3" />
```

Note: The value types for Viewpoint properties include the following:

- **`Boolean`** Indicates off or on; value “0” or “1.”
- **`String`** String of characters.
- **`Integer`** Whole numbers.
- **`Real`** Floating point numbers.
- **`Point3d`** Set of three floating point numbers.
- **`Point2d`** Set of two floating point numbers.
- **`Quaternion`** Set of four floating point numbers, the first three specifying the 3D axes of rotation, the other being the cosine of an angle rotation.
- **`Rect`** Set of four floating point numbers.
- **`Matrix`** Set of 12 floating point numbers.
- **`Unknown`** Direct pointer to objects of any type (the object’s type is not specified). The `Unknown` value type is NOT the same as “undefined.” “Undefined” means the type is not known because, in most cases, the property does not exist.

New DOM Syntax

The DOM syntax for Viewpoint Media Player releases prior to 3.0.14 uses the convention of double colons (`::`) to distinguish the object name from the property name.

The new DOM syntax employs the period (`.`) to distinguish the object from the property name. For example:

```
<MTSSetProperty Target="MTSInstance.cube.loc_" Value="1 2 3" />
```

This new DOM syntax convention harmonizes with the pre-existing syntax of the period (`.`) between the type of the object and its name. It also harmonizes Viewpoint content authoring

syntax with all other DOM syntax available in JavaScript, C++, Flash ActionScript, and so forth.

Notes:

- The DOM syntax of pre-3.0.14 release versions remains valid to support compatibility with existing content.
- To use the new DOM functionalities, the 3.0.14 SceneComponent component is required.

Properties of Properties (Sub-Properties)

A major advantage of DOM syntax is how it allows you to define a path to information.

For example, `MTSInstance` objects have a property called `Geometry`, which corresponds to the `MTSGeometry` tag in MTX code. The `Geometry` property has properties of its own, including `Vertices`.

```
<!--Here, we access the Vertices property of the Geometry property of
the MTSInstance named "cube." -->

MTSInstance.cube.Geometry.Vertices
```

Also, the basic value types, such as `Point3d`, `Point2d`, `Quaternion`, `Rect`, now have sub-properties. For example, the `MTSInstance` property, `Translation` (or even `loc_`), now has the following sub-properties: `x`, `y`, and `z`. Therefore, to modify only an object's `x` translation value, specify the following path:

```
MTSInstance.cube.Translation.x
```

The following table presents the sub-properties of the basic value types:

Value Type	Sub-Properties
`Point3d`	`x, y, z`
`Point2d`	`x, y`
`Quaternion`	`qx, qy, qz, qs`
`Rect`	`left, top, right, bottom, x` (same as `left`), `y` (same as `top`), `width` (same as `right -left`), `height` (same as `bottom - top`)
`Matrix`	`Rotation, Translation, Scale, Shear`

Sub-Property Notes:

- The value type for all of the sub-properties is `Real` EXCEPT the `Matrix` type sub-properties, `Rotation`, `Translation`, `Scale`, and `Shear`, which are `Point3d`.
- For the `Quaternion` value type, the sub-properties `x`, `y`, and `z` also work. Viewpoint Media Player interprets these sub-properties as Euler angles.

Usage Notes:

The new DOM syntax is compatible with previously released four-letter properties. For example, the following is a valid DOM path:

```
MTSInstance.cube.loc_.x
```

The `Translation` property was added before the DOM syntax was enabled (in other words, before it was possible to declare: `Matrix.Translation`.). As a result, the following two path declarations are possible, with both pointing to the same information:

1. `MTSInstance.cube.Translation`
2. `MTSInstance.cube.Matrix.Translation`

In effect, the first declaration is a shortcut of the second declaration. However, in some cases, only the second declaration is available. For example, these declarations do not have a shortcut version:

- `InvMatrix.Translation`
- `InvMatrix.Rotation`
- `WorldMatrix.Translation`

In summary, now it is possible to request the following information:

```
<!--This declaration is the path to the x value of the rotation
specified by the world matrix of the MTSInstance named "cube". -->

MTSInstance.cube.WorldMatrix.Rotation.x
```

Array Properties

Some properties do not correspond to one value but to a set of indexed values. These property types are called array properties. For example, the `MTSGeometry` property, `Vertices`, is an array property. The `Vertices` array property accesses all vertex coordinates of a geometry.

Note: Each vertex has 3D coordinates (value type: `Point3d`); the `Vertices` property is an array of the `Point3d` value type.

To specify the element of the array to be accessed:

1. Specify the path to the property.
2. Immediately following the name of the specified property, enter the index to be accessed between brackets: `[' ']`
   All arrays are zero-based.

For example, to access the first vertex of the `MTSGeometry` named "geom.", specify the following DOM path:

```
<!--Here, this DOM path acts as if it were a simple property of the
value type, Point3d.

MTSGeometry.geom.Vertices[0]
```

Due to the fact that the property's value type is `Point3d`, you can request more specific information, as previously explained. For example:

```
MTSGeometry.geom.Vertices[0].y
```

Usage Notes:

When setting the value of an element of an array property that does not exist, the declaration fails. For example, if the array has only five elements and in the declaration you set the value of the tenth element, the declaration fails because the tenth element does not exist.

Continuing this example, note that when setting the value of the *sixth* element (the non-existing element that sequentially follows the final existing element), Viewpoint Media Player automatically inserts the sixth element into the array and the declaration does not fail.

Usage Notes Addendum:

Request (or set) the number of elements in an array using the sub-property, `count` (value type, `mts_int`). For example:

```
MTSGeometry.geom.Vertices.count
```

Automatic Path Resolution

Automatic path resolution has been a DOM feature from the time of Viewpoint Media Player's inception. Prior to release 3.0.14, in declarations specifying parameters for an action, the DOM tried to resolve the specifying path. If the path failed, Viewpoint Media Player interpreted the specified parameters as the value type, `string`.

Effective release 3.0.14, path resolution has been enhanced to handle the extended syntax of the DOM. The following usage examples for path resolution demonstrate these two advantages of the enhanced DOM syntax:

1. `MTSAction` can be used like a function in JavaScript and C++: it passes parameters, thereby generalizing the action.
   Now, there's no need to create a specific action for every scenario, which greatly enhances Viewpoint Media Player's scriptability.
2. Now, `MTSAssignProperty` does exactly the same thing as `MTSCopyProperty`, making the latter obsolete.
   For example, the following two declarations do the same thing:
   - `<MTSAssignProperty Target="cube.Translation" Value="sphere.Translation" />`
   - `<MTSCopyProperty Dest="cube.Translation" Src="sphere.Translation" />`

To understand the advantages of DOM path resolution, review the following three examples:

Example 1: In this example, "`cube.Translation`" is a valid DOM path while "`zoubida`" is not. The result is that the valid path is resolved and the invalid path is assigned the value type, `String`.

```
<!--Here, "myPara1" directly points to the property, Translation,
which pertains to the object named "cube." -->

<MyAction myPara1="cube.Translation" myPara2="zoubida" />
```

Example 2: The following declaration specifies the action, "MyAction," which sets the value "0 0 0" to the target stored in the parameter, "myPara1":

```
<MTSAction Name="MyAction" >
    <MTSSetProperty Target="myPara1" Value="0 0 0" />
</MTSAction>
```

To set the translation of the object, "cube," to the position "0 0 0," call the action as follows:

```
<MyAction myPara1="cube.Translation" />
```

However, if you were to call the action, `<MyAction myPara1="zoubida" />`, the action would fail because "zoubida" is just a string.

Example 3: The following declaration specifies the action, "MyAction," which sets the value "0 0 0" to the Translation of the object stored in the parameter, "myPara":

```
<MTSAction Name="MyAction" >
    <MTSSetProperty Target="myPara.Translation" Value="0 0 0" />
</MTSAction>
```

To set the Translation property of the object, "cube," to the position, "0 0 0," call the action as follows:

```
<MyAction myPara="cube" />
```

To set the translation of the object, "sphere," to the position, "0 0 0," call the action:

```
<MyAction myPara="sphere" />
```

To set the Translation of the first child of the MTSInstance named, "cube," to the position, "0 0 0," call the action as follows:

```
<MyAction myPara="MTSInstance.cube.SceneChilds[0]" />
```

Functions

Objects have properties and functionalities (functions) that can be accessed using a DOM path. An object functionality (or function) is a specific action that a specific object knows how to execute.

Unlike properties, functions do not store a state of the object, but instead trigger an action on the object. Currently, very few objects execute these built-in functions.

To call a function on an object, use the action, VETDispatchCall, as follows:

```
<!--This example calls the function, ClearScene, on the object named
MTSBaseComponent. -->.

<VETDispatchCall Function="MTSBaseComponent::ClearScene()" />
```

Usage Notes:

Currently, contrary to the new DOM syntax convention, function calls still are declared using the double-colon syntax (::) instead of the period (.).

Accessing Properties Using the DOM Syntax

The extended DOM syntax can be used in the following ways to access property values:

Using MTSSetProperty and MTSAssignProperty

Set property values using an action specified by `MTSSetProperty` and `MTSAssignProperty` as follows:

- `<MTSSetProperty Target="MTSGeometry.geom.Vertices[0]" Value="0 0 0" />`
- `<MTSAssignProperty Target="MTSGeometry.geom.Vertices[0]" Value="0 0 0" />`

When using `MTSSetProperty` to set property values via the DOM syntax, if the specified path does not exist, the DOM automatically creates the property, if possible. For example:

- In the following declaration, the DOM automatically creates the property, "toto," for the object named "cube," because this property does not exist:
  `<MTSSetProperty Target="MTSInstance.cube.toto" Value="0 0 0" />`
- In the following declaration, the DOM automatically creates the property, “toto,” for the geometry of the instance named, “cube,” because this property does not exist:
  `<MTSSetProperty Target="MTSInstance.cube.Geometry.toto" Value="0 0 0"/>`
- In the following declaration, the DOM *cannot* automatically create the property, “toto,” because the property, “vine,” does not exist either:
  `<MTSSetProperty Target="MTSInstance.cube.vine.toto" Value="0 0 0" />`

Furthermore when setting property values with `MTSSetProperty`, the value type of an automatically created property mirrors the type of the value specified in the declaration. For example:

- In the following declaration, the DOM creates a property named "toto" on the object named, "cube." The value type of “toto” automatically is `Point3d` and its value is initialized to the value of the translation of the object, "sphere":

  ```
  <MTSSetProperty Target="MTSInstance.cube.toto"
   Value="MTSInstance.sphere.Translation" />
  ```

- In the following declaration, the DOM creates a property named “toto” (its value type automatically is `Unknown`), pointing directly to the object named, “sphere”:

  ```
  <!--Once this action is executed, the path, “MTSInstance.cube.toto”
  points on the same interaction as the path, “MTSInstance.sphere”-->

  <MTSSetProperty Target="MTSInstance.cube.toto"
   Value="MTSInstance.sphere" />

  <!--So, to access the property, Translation, of the objects, “cube”
  and “sphere,” the following two declarations are valid: -->
  ```

  - `MTSInstance.cube.toto.Translation`
  - `MTSInstance.sphere.Translation`

Using a DOM path as the target of an animator

Use the `Path` attribute instead of the `Name` and `Property` attributes, demonstrated as follows:

```
<MTSTimeElem Type="Keyframe" >
   <Target Path="MTSInstance.cube.Translation.x" Timeline="T1" />
   <Time> 0 1 2 </Time>
   <Timeline Name="T1" Type="1D">[0] [1] [2] </Timeline>
</MTSTimeElem>
```

With this new syntax, you no longer have to declare the targeted object before the animator declaration. The resolution of the DOM path is done during the first rendering pass (instead of of during parsing).

Using DOM with JavaScript

The following declaration demonstrates how to use a DOM path in JavaScript:

```
<!--This declaration corresponds to the path,
“cube.Matrix,Translation.” -->

vmp.SetProperty('cube', 'Matrix.Translation', '0 0 0');
```

Using XML tags

With `<MTSScene Version="311">` or higher, properties can be used as tags. This functionality voids the distinction between property names and tag names in MTX code.

The properties can be specified as XML attributes or as child nodes. The following declarations demonstrate the XML specification methods, which accomplish the same outcome:

- `<VETOrbitManipulator Translation="1 0 0" />`
- ```
  <VETOrbitManipulator>
   <Translation>1 0 0</Translation>
  </VETOrbitManipulator>
  ```
- ```
  <VETOrbitManipulator Translation.x="1" Translation.y=”0”
  Translation.z=”0” />
  ```

In XML, it is possible to specify array properties to set the values of the properties of an object by using any of the following syntax options, though it is *impossible* to set array properties as attributes of a node:

- In this declaration, the array property, `Vertices`, is set to the count of three elements, which are then set to values specified:
  ```
  <MTSGeometry>
  <Vertices>[0 0 0][1 0 0][0 1 0]</Vertices>
  </MTSGeometry>
  ```
- In this declaration, the array property is specified element-by-element, each one being added to the array:
  ```
  <MTSGeometry>
  <Vertices>0 0 0</Vertices>
  <Vertices>1 0 0</Vertices>
  <Vertices>0 1 0</Vertices>
  </MTSGeometry>
  ```

VETOrbitManipulator

The `VETOrbitManipulator` animator is a new feature of Viewpoint Media Player, making it possible to manipulate any object in a scene that has a matrix property type, such as `VETCamera`. The most common application of this feature is to provide interaction to `VETCamera`, although the `VETOrbitManipulator` animator can be applied to any instance in a scene.

VETOrbitManipulator and VETCamera

`VETOrbitManipulator` automatically controls `VETCamera` when the following conditions are met:

- Scene version must equal or exceed 314 (`<MTSScene Version="314" >`).
- There is NO declaration of `MTSCamera` in the scene.
- There IS a declaration of `VETCamera`.

When these conditions are met, it is unnecessary to specify `VETOrbitManipulator` in MTX code unless you want to override the `VETOrbitManipulator` default values. The default `VETOrbitManipulator` animator is named, "CameraManipulator." The default behavior for `VETOrbitManipulator` mirrors the default settings for `MTSCamera` (placing the camera at a distance of 4 units from the center of the scene with no rotation).

To customize the settings for VETOrbitManipulator:

1. In the .mtx file, declare the `VETOrbitManipulator` animator.
2. Name the animator, "CameraManipulator."
3. Define the animator's attributes according to your specifications.
   This declaration automatically overrides the default settings:

```
<!--This declaration positions the camera 8 units from the center
of the scene and enables orbit rotation of 60 degrees around the y
axis. -->

<VETOrbitManipulator Name="CameraManipulator" Distance="8"
 Orbit="0 60 0" />
```

> **Note:** The `VETOrbitManipulator` tag is also an action, meaning it can be called from an event, allowing you to control the target position when receiving a `Mousemove` event.

The VETOrbitManipulator Tags and Properties

Tag and Property	Description
`<VETOrbitManipulator>`	Manipulates any object in a scene that has a matrix property type, such as `VETCamera`.
`Name`	Specifies the name of the `VETOrbitManipulator` animator. When used with `VETCamera` to override default `VETOrbitManipulator` settings, declare: `Name="CameraManipulator"`.
`On`	Determines whether the animator is on or off. If On, it runs when the page is loaded. If Off, it has to be triggered by a user action.
`Targets`	Specifies the targeted object properties using a DOM path. This property is an array property.
`Pivot`	Changes the pivot point (anchor position) of the targeted object in a scene.
`Rotation`	Determines how far the targeted object rotates around a neutral (zero) position using a quaternion, which allows it to be animated without rendering glitches.
`Translation`	Determines the position of the targeted object along the x, y, and z axes.
`Scale`	Changes the scale of the targeted object.
`Distance`	Defines the targeted object's distance from the pivot. Distance is the exact opposite of `Translation.z`.
`Orbit`	Determines how far the targeted object rotates around a neutral (zero) position using Euler angles, emulating the `MTSCamera` property, `rot_`. To solve the gimble lock problems, we recommend using the `Rotation` property, not `Orbit`.
`Pitch`	Determines how far the targeted object rotates around a neutral (zero) position in relation to the x axis.
`Yaw`	Determines how far the targeted object rotates around a neutral (zero) position in relation to the y axis.

Tag and Property	Description
`Roll`	Determines how far the targeted object rotates around a neutral (zero) position in relation to the z axis.
`Matrix`	Specifies the position, rotation and scale of the targeted object.
`InvMatrix`	Specifies the inverse position, rotation, and scale of the targeted object.

Tag and Property	Description
`LookAt`	Sets the pivot point of the targeted object and computes its new `Rotation` so that it focuses on the new pivot point without actually changing the object's position.
`FromPoint`	Sets the position of the targeted object and computes its new `Rotation` so that it keeps focusing on the same pivot point.
`RotationMin`	Sets the minimum limits for the targeted object's rotation around the center of the scene.
`RotationMax`	Sets the maximum limits for the targeted object's rotation around the center of the scene.
`TranslationMin`	Sets the minimum limits for the targeted object's position along the x, y, and z axes.
`TranslationMax`	Sets the maximum limits for the targeted object's position along the x, y, and z axes.
`DistanceMax`	Defines the maximum limit of the targeted object's distance from the pivot point.
`TranslationScale`	Controls the mouse scale so that the mouse has more or less effect on the targeted object's translation.
`RotationScale`	Controls the mouse scale so that the mouse has more or less effect on the targeted object's rotation.
`RotationInertia`	Specifies the rotation inertia resulting from user mouse interaction with the targeted object.
`TranslationInertia`	Specifies the translation inertia resulting from user mouse interaction with the targeted object.

Tag and Property	Description
`MaximumRotateVelocity`	Specifies the maximum rotation velocity in degrees per second resulting from user mouse interaction with a targeted object, such as how fast a cube can spin around the x axis.
`MaximumTranslateVelocity`	Specifies the maximum translation velocity resulting from user mouse interaction with the targeted object, such as how fast a cube moves along the x axis.
`AutoAdaptMatrix`	Detects when the matrix has been changed by a different actor than the `VETOrbitManipulator` animator itself. Once changes are detected it automatically adapts the internal variables to the new matrix.

VETOrbitManipulator Sample MTX Code

The following three MTX code sample demonstrates how to use `VETOrbitManipulator`: using default settings and targeting `VETCamera`, applying customized settings to `VETCamera`, and defining a customized mouse event handler for manipulating a cube in a scene.

```
<!--Example 1: -->
<!-- Here, an animation animates the default VETOrbitManipulator
feature, which animates the camera. -->

<MTSScene Version="314">
    <VETCamera Name="MyCamera"  />

    <MTSTimeElem Type="Keyframe" On="1">
        <Target Path="CameraManipulator.Rotation.x" Timeline="T0"/>
        <Time> 2 3 </Time>
        <Timeline Name="T0" Type="1D"> [0] [90] </Timeline>
    </MTSTimeElem>
</MTSScene>

<!--Example 2: -->
<!-- Here, the declared VETOrbitManipulator overrides the default
VETOrbitManipulator values. -->

<MTSScene Version="314">
    <VETCamera />
    <VETOrbitManipulator Name="CameraManipulator" Pivot="0 1 0" Yaw="-
1" Scale="1 0 1" />
</MTSScene>

<!--Example 3:-->
<!--Here, we define our own mouse events handler for manipulating a
cube in the scene: -->

<MTSScene Version="314">
    <MTSInstance Name=”cube” />
        <MTSGeometry Type=”MTSCube” />
    </MTSInstance>

    <VETOrbitManipulator Name="MyManipulator" />
        <Targets>cube.InvMatrix</Targets>
    </VETOrbitManipulator>

    <MTSInteractor Name="BothButtonsInteractor"
```

```
  StartState="NoButtonDown" IgnoredHandle="1" >
     <MTSHandle Event="MouseLeftDown" StartState="NoButtonDown"
      EndState="LeftButtonDown" doInertia="0"
      Action="MyManipulator"/>
     <MTSHandle Event="MouseLeftDown" StartState="RightButtonDown"
      EndState="BothButtonsDown" Action="MyManipulator"/>
     <MTSHandle Event="MouseLeftUp" StartState="LeftButtonDown"
      EndState="NoButtonDown" doInertia="1" Action="MyManipulator"/>
     <MTSHandle Event="MouseLeftUp" StartState="BothButtonsDown"
      EndState="RightButtonDown"   Action="MyManipulator"/>

     <MTSHandle Event="MouseRightDown" StartState="NoButtonDown"
      EndState="RightButtonDown" doInertia="0"
      Action="MyManipulator"/>
     <MTSHandle Event="MouseRightDown" StartState="LeftButtonDown"
      EndState="BothButtonsDown" Action="MyManipulator"/>
     <MTSHandle Event="MouseRightUp" StartState="RightButtonDown"
      EndState="NoButtonDown" doInertia="1" Action="MyManipulator"/>
     <MTSHandle Event="MouseRightUp" StartState="BothButtonsDown"
      EndState="LeftButtonDown" Action="MyManipulator"/>

     <MTSHandle Event="MouseDrag" StartState="LeftButtonDown"
      deltaPitch="MTSEvent::_dy_" deltaYaw="MTSEvent::_dx_"
      Action="MyManipulator" />
     <MTSHandle Event="MouseDrag" StartState="RightButtonDown"
      deltaZoom="MTSEvent::_dy_" Action="MyManipulator" />
     <MTSHandle Event="MouseDrag" StartState="BothButtonsDown"
      deltaPanX="MTSEvent::_dx_" deltaPanY="MTSEvent::_dy_"
      Action="MyManipulator" />
</MTSInteractor>
```

VETSequencer

The `VETSequencer` animator is a new feature of Viewpoint Media Player, making it possible to synchronize animations that do not run for the same length of time.

A Viewpoint animation's time is dictated from the parent animator. Now, via `VETSequencer`, you can specify a list animators to which you want to synchronize, the last animator specified being the highest priority. The key is to specify the list of animators in the desired priority order.

For example, this feature allows a Flash movie to be synchronized with several consecutive video animators. To utilize `VETSequencer` in content, familiarize yourself with the Viewpoint XML tags and properties: `VETSequencer` and `Synchronizers`.

The VETSequencer and Synchronizers Tags and Properties

Tag and Property	Description
`<VETSequencer>`	Unlike many Viewpoint animators, `VETSequencer` is not an `MTSTimeElem` element type (such as Flash or SVG). The VETSequencer feature is declared in MTX code as follows: `<VETSequencer Name="coco"  On="1">`
`Synchronizers`	Designates an animator for synchronization, as demonstrated below. This is an array property.

VETSequencer Sample MTX Code

The following MTX code sample demonstrates a simple scene using `VETSequencer`.

```
<!--The following two MTSTimeElem declarations serve as sample
animations. -->

<MTSTimeElem Name="Bob" On="0" />

<MTSTimeElem Name="Bob2" On="0" />

<!--Here, declare the VETSequencer tag as you would any Viewpoint
animator and specify the list of animations to synchronize. -->

<VETSequencer Name="coco"  On="1" >
    <Synchronizers>Bob</Synchronizers>
    <Synchronizers>Bob2</Synchronizers>

<!--Here, declare all the animations (as children of VETSequencer) to
synchronize. -->

<MTSTimeElem Type="Keyframe" On="1" Clamp="0">
    <Target Path="cube.Translation" Timeline="T0"/>
    <Time> 0 1 2 3 4 5 6 </Time>
    <Timeline Name="T0" Type="3D"> [0 0 0][1 0 0][1 0 1][0 0 1][-1 0
     1][-1 0 0][0 0 0]</Timeline>
</MTSTimeElem>
</VETSequencer>
```

Usage Notes:

To better demonstrate how `VETSequencer` works in the context of the sample code above, note that when:

- The animator named “Bob” is ON and the animator “Bob2” is OFF, the animation synchronizes to “Bob.”
- The animator named “Bob” is OFF and the animator “Bob2” is ON, the animation synchronizes to “Bob2.”
- The animator named “Bob” is ON and the animator “Bob2” is ON, the animation synchronizes to “Bob2.”
- The animator “Bob” is OFF and “Bob2” is OFF, the animation synchronizes to the parent of the `VETSequencer` declaration.

VETStreamCase

`VETStreamCase` is a new feature of Viewpoint Media Player, introducing functionality that enables you to dynamically choose the MTX child node to execute based on the type of media file streamed from a host server, even if the exact file type is not known.

In effect, this feature establishes conditional logic within a scene whereby several `VETStreamCase` scenarios can be declared, each one specifying a particular file type and containing a particular nested MTX child node. If the file type is found, Viewpoint Media Player executes the child node. If the file type is not found, `VETStreamCase` is ignored as is its child node.

The VETStreamCase Tags and Properties

Declare `<VETStreamCase>` when the file type sought is unknown (the file path must be known).

Tag and Property	Description
`<VETStreamCase>`	Dynamically executes a nested MTX child node of the `VETStreamCase` animator based on the type of media file it loads from a host server.
`Path`	Specifies the path for the media file being loaded.
`Signature`	Defines a known type of file, including “JPEG,” “GIF,” “MTZ,” “MTX,” and “VMPVIDEO,” to be streamed from a host server.
`ByteSignature`	Defines the file signature (type) that must pertain to a byte location within a file to be streamed from a host server.
`Offset`	Specifies the byte location within a file where Viewpoint Media Player searches for the file's specified signature (type).

VETStreamCase Sample MTX Code

The following sample MTX code demonstrates how to use the `VETStreamCase` tag suite.

```
<!--EXAMPLE 1: Using the Signature tag to find a known file. -->
<!--Here, the VETStreamCase declaration directs Viewpoint Media Player
to go the the specified path to find the .jpg file there. If the file
is there and is of type “JPEG,” the player executes the nested
<MTSTimeElem> code. -->

<VETStreamCase Signature="JPEG" Path="somefile.jpg">
   <MTSTimeElem Name="anyjpg"  Type="MTSImageStream"  On="1"
    Path="somefile.jpg" >
      <Target Name="MTSTexture.anycontent" />
   </MTSTimeElem>
</VETStreamCase>

<!--EXAMPLE 2: Using the ByteSignature and Offset attributes to find an
unknown file (or not accessible via the Signature tag). -->
<!--In this example, the ByteSignature and Offset attributes are used
in place of Signature to find the specified signature type, “FWS”(or
.swf file), according to the specified byte location given in the
Offset value, "0” (or the first byte of the file). If it finds it,
Viewpoint Media Player streams the file and executes the nested
animation. If not, it ignores this declaration and goes to the next.-->
```

```
<VETStreamCase ByteSignature="FWS" Offset="0" Path="mypath.swf">
    <MTSTimeElem Name="anyjpg"  Type="MTSImageStream"  On="1"
     Path="somefile.jpg" >
        <Target Name="MTSTexture.anycontent" />
    </MTSTimeElem>
</VETStreamCase>
```

MTSBaseComponent Functions

Unlike properties, functions do not store a state of an object, but instead trigger an action on the object. Currently, very few objects feature these built-in functions. For more information on functions, see "Functions".

Most object functions relate to existing objects, such as `MTSBaseComponent`, and are called via the `<VETDispatchCall Function>` command.

Note: The `<VETDispatchCall Function>` tag can be researched at Viewpoint XML Reference Guide.

Viewpoint Media Player 3.0.14 introduces the following function.

RemoveObject

This `MTSBaseComponent` object function are an enhancement in scene or object "clearing" functionality, such as the previously released `ClearScene` function.

Function	Description
`RemoveObject`	Removes a specified object from a scene and memory (RAM) when called.

The following sample MTX code demonstrates how to use this function:

```
<!--EXAMPLE 1: Here, the RemoveObject function removes the "MyObject"
object from the scene. "MyObject" can be any object defined in the
scene. -->

<VETDispatchCall Function="MTSBaseComponent::RemoveObject(MyObject)"/>

<!--EXAMPLE 2: Here, the RemoveObject function removes the first child
("MyChild") of the instance, "MyInstance," from the scene -->

<MTSInstance Name="MyInstance" >
    <MTSInstance Name="MyChild" />
</MTSInstance>

<VETDispatchCall
Function="MTSBaseComponent::RemoveObject(MyInstance.SceneChilds[0])"/>
```

VETRandomGenerator Tags and Properties

These tags and properties belong to the existing `VETRandomGenerator` functionality suite, which enables random numbers or strings to be generated according to specified ranges. Two new tags and properties have been added to `VETRandomGenerator`.

For more information on the `VETRandomGenerator` function suite, see "VETRandomGenerator"

Tag and Property	Description
`Strings`	Defines all of the strings that can be returned in random generation. This property is an array property.
`RandomString`	Returns a string defined for random generation. This property is a read-only property.

The following sample MTX code demonstrates how to use the new `VETRandomGenerator` tags and properties:

```
<!--Here, the Strings property defines the array of strings that can be
returned. -->

<VETRandomGenerator Name="myName" >
    <Strings>[toby][toto][toby2] [toto2] </Strings>
</VETRandomGenerator>

<!--Here, the RandomString read-only property is called to select one
of the above strings. -->

<MTSInteractor>
    <MTSHandle Event="MouseLeftDown" Action="MTSJavaScript"
     Func="alert(myName::RandomString);" />
</MTSInteractor>
```

Viewpoint Media Player Events

Viewpoint Media Player 3.0.14 introduces a host of new events and enhanced event functionality.

Collecting Events by Their Generic Names

The DOM syntax introduces a new way to collect events by their generic name, demonstrated here:

```
<!--The new way you can collect events does not require you to specify
the full event name. -->
<!--For example: <MTSHandle Event="MTSLoadDone:EventName">. Now, the
generic event name is sufficient. Of course, if there are several
objects in the scene that can send the event, MTSLoadEvent, this
MTSInteractor declaration will receive all of them. -->

<MTSInteractor />
    <MTSHandle Event="MTSLoadDone" Action="MTSJavaScript"
     Func="alert();" />
</MTSInteractor/>
```

MTSEvent Read-Only Properties

Viewpoint Media Player 3.0.14 introduces the following two new `MTSEvent` read-only properties to help determine the object that fires an event and the name of the event fired.

Read-Only Property	Description
`Sender`	Accesses and returns the object that triggered and sent an event. `Sender` is a read-only property.
`EventName`	Returns the generic name of the event that was triggered and sent. `EventName` is a read-only property.

The following MTX code samples demonstrate how to use these tags and properties.

```
<!--EXAMPLE 1: Sender -->
<!--Access the event-sender object as follows. -->

<MTSInteractor>
    <MTSHandle Event="MTSLoadDone:Blob" Action="MTSSetProperty"
     Target="MTSEvent.Sender.runs" Value="0" />
</MTSInteractor>

<!-- Note that this declaration pops up a dialog box with the name of
the object that sent the event. -->

<MTSInteractor>
    <MTSHandle Event="MTSLoadDone" Action="MTSJavaScript"
     Func="alert(MTSEvent.Sender.name);" />
</MTSInteractor>

<!--EXAMPLE 2: EventName -->
<!-- Note that this declaration pops up a dialog box with the generic
name. -->

<MTSInteractor>
    <MTSHandle Event="MTSLoadDone:myanimator" Action="MTSJavaScript"
     Func="alert(MTSEvent.EventName);" />
</MTSInteractor>
```

Usage Notes:

- **`Sender`** This property's value type is `Unknown`.
- **`EventName`** This property's value type is `String`.

Enhanced Event Functionality: KeyUp and KeyDown

The `KeyUp` and `KeyDown` events already exist as part of the Viewpoint event suite, but have been significantly enhanced, as demonstrated in the following four MTX code samples:

```
<!--EXAMPLE 1: KeyUp and KeyDown -->
<!--KeyUp and KeyDown include new features, as described here, that
ensures that each time the user presses down or releases a key, a
dialog box launches displaying the affected key.  -->

<MTSInteractor>
    <MTSHandle Event="KeyDown" Action="MTSJavaScript"
     Func="alert(MTSEvent.Sender.name);" />
    <MTSHandle Event="KeyUp" Action="MTSJavaScript"
     Func="alert(MTSEvent.Sender.name);" />
```

```
</MTSInteractor>

<!--EXAMPLE 2: Filtering keys -->
<!--Furthermore in relation to the new KeyUp and KeyDown features,
filtering a specific key is possible, as demonstrated below catching
only the 'A' key. This works for non-English keyboards, non-PC
keyboards, as well as non-alphanumeric keys, such as the arrows (LEFT,
RIGHT, UP, DOWN). The key location mapping, though, is based on the PC-
friendly keyboard, so, for example, getting the Apple key brings up the
WINDOWS key message corresponding to a PC keyboard. -->

<MTSInteractor>
    <MTSHandle Event="KeyDown:A" Action="MTSJavaScript" Func="alert('A
     pressed');" />
</MTSInteractor>

<!--EXAMPLE 3: Non-Alphanumeric keys -->
<!--Use the following declaration to get the code name for the non-
alphanumeric keys: PAGEUP, PAGEDOWN, HOME, END, INSERT, DELETE, BACK,
CAPSLOCK, ESCAPE, RETURN, SPACE, CONTROL, SHIFT, NUMLOCK, NUMPAD0,
NUMPAD1, etc., DECIMAL, SCROLLLOCK, PAUSE, WINDOWS, CONTEXTMENU,
DIVIDE, MULTIPLY, SUBSTRACT, ADD, EQUAL, SINGLEQUOTE, DASH, BACKSLASH,
CLOSEBRACKET, OPENBRACKET, SEMICOLON, APOSTROPHE, COMMA, PERIOD,
SLASH, BACKSLASH. -->

<MTSInteractor>
    <MTSHandle Event="KeyDown" Action="MTSJavaScript"
     Func="alert(MTSEvent.Sender.name);" />
</MTSInteractor>

<!EXAMPLE 4: Ascii code -->
<!--Lastly, 3.0.14 allows you to request the Ascii code for the
combination of key that has just been pressed,
using property "Ascii" (type String) on the sender of the event. -->

<MTSInteractor>
    <MTSHandle Event="KeyDown" Action="MTSJavaScript"
     Func="alert(MTSEvent.Sender.Ascii);" />
</MTSInteractor>

<!--EXAMPLE 5: Passing key events to a .swf file -->

<MTSTimeElem Type=”SWFView” Name=”MySWF” On=”1” Path=”myPath.swf” />

<MTSInteractor>
    <MTSHandle Event="KeyDown" Action="MySWF" />
</MTSInteractor>
```

MTSEvent Picking-Specific, Read-Only Properties

Viewpoint Media Player 3.0.14 introduces the following five new `MTSEvent` properties to help determine the UV and 3D coordinates of picked textures and points, indices of geometries, materials, and instances when a user clicks them.

Read-Only Property	Description
`TxtCoords`	Gets the texture point coordinates of the picked texture.
`WorldPoint`	Gets the picked 3D point coordinates of the scene.
`UV`	Gets the UV coordinates in the picked texture.

Read-Only Property	Description
`FaceIndex`	Gets the face index of the picked geometry.
`Point`	Gets the 3D coordinates of the picked point relative to the picked object.
`Material`	Gets the picked material.
`Instance`	Gets the picked instance.

The following MTX code sample demonstrates how to use each one of the listed picked-specific properties. This same sample applies to all of the listed properties; simply substitute one for another.

```
<!--Here, get UV with an interactor. -->

<MTSInteractor Name="myInteractor">
   <Target Name="Simple_0"/>
   <MTSHandle Event="MouseLeftClick" Action="MTSJavaScript"
    Func="alert(MTSEvent.UV)"/>
</MTSInteractor>
```

The MTSAnimatorAvailable Event

Viewpoint Media Player 3.0.14 introduces a new event, `MTSAnimatorAvailable`, that is fired when the targeted animator has downloaded. The event is fired even if the animator is already present on the user's machine. This allows you to safely wait for the event without worrying whether the animator will be downloaded or is already present. Once you receive the event, the animator is present.

This event can be extremely useful if content requires the presence of a specific animator in order to execute correctly.

Video Events

For information on the new Viewpoint Video events, see "Video Events".

MTSTimeElem Tags and Properties

Viewpoint Media Player 3.0.14 introduces two new `MTSTimeElem` tags and properties.

Tag and Property	Description
`Targets`	This new array property makes it possible to: • Allow multiple targets to be specified in one operation. • Change the property at runtime, meaning that this property can change the property being animated by another property.

Tag and Property	Description
`DownloadingAnimator`	Returns the status of an animator's presence in a scene as either downloaded or in the process of being downloaded. If the animator is already present, this property does not exist. Use the `MTSHandle Event, MTSAnimatorAvailable`, instead of `DownloadingAnimator`, because the `DownloadingAnimator` property may or may not exist.

The following MTX code samples demonstrate how to use these new tags and properties.

```
<!--EXAMPLE 1: Targets -->
<!--Setting two targets in one operation. -->

<VETOrbitManipulator >
   <Targets>[myName.pogodo][myName2.pogodo]</Targets>
</VETOrbitManipulator >

<!--EXAMPLE 2: Targets (multiple)-->
<!--Setting two targets in one operation. -->

<VETOrbitManipulator >
   <Targets>myName.pogodo</Targets>
   <Targets>myName2.pogodo</Targets>
</VETOrbitManipulator >
```

The Propagate Tag and Property

Viewpoint Media Player 3.0.14 introduces the tag and property, `Propagate`, which can be used by `<MTSSetProperty>` and `<MTSAssignProperty>` to propagate a property value or setting across the entire scene hierarchy. For known limitations with this tag, see "Propagating an Action Using the DOM".

Tag and Property	Description
`Propagate`	Propagates a property along the scene hierarchy.

The following MTX code samples demonstrate how to use both `<MTSSetProperty Propagate>` and `<MTSAssignProperty Propagate>`.

```
<!--EXAMPLE 1: <MTSSetProperty> -->
<!-- Here, the opac property, which sets the opacity of the instance,
"MyInst," is propagated to the instance's children. -->

<MTSSetProperty Target="MyInst::opac" Value="0.5" Propagate="1" />

<!--EXAMPLE 2: <MTSAssignProperty> -->
<!--Here, the opac property, which sets the opacity of the instance,
"BobInst,"  is propagated to the instance's children: -->

<MTSAssignProperty Target="BobInst::opac" Value="0.5" Propagate="1" />

<!--EXAMPLE 3: Using from JavaScript -->
```

```
vmp.SetProperty('MyInst','opac',0.5,'mts_real','propagate');
```

Note: To use the `Propagate` feature from JavaScript, use the MTS3Interface version 4.0.8.2 or higher.

The IgnoreActionError Tag and Property

The newly introduced `IgnoreActionError` tag and property can be used to continue executing a chain of actions even if one of them fails.

Tag and Property	Description
`IgnoreActionError`	When calling an action, `IgnoreActionError="1"` specifies that even if the action fails, it should continue executing the remaining actions.

The following MTX code sample demonstrates how to use this tag and property.

```
<!--Here, the Target value, “myobject” does not exist, by default
failing the action. However, IgnoreActionError is set to “1” to ensure
that all remaining actions execute. -->

<MTSSetProperty Target="myobject.object" Value="3"
IgnoreActionError="1" />
```

MTSInstance Tags and Properties

Viewpoint Media Player 3.0.14 introduces several new `MTSInstance` tags and properties.

Tag and Property	Description
`Materials`	Specifies the material to use in a scene at the scene's runtime. This is an array property. For information on the three properties that correspond to Materials, see “MTSMaterial Tags and Properties”.
`Geometry`	Sets or gets the geometry object associated with an instance. The most useful feature of this property is that it can change a property on the geometry of an instance without needing to know the name for the geometry itself.
`RenderStyle`	Defines the render mode for a particular object in the scene. The render mode describes how the diffuse color, diffuse texture, and the environmental lightmap combines to create the impression of a lit surface.

Tag and Property	Description
`BackFaceDirection`	Determines the facing direction of an object's polygons. Use this tag to flip polygons by inverting the direction they face.
`BackFaceCulling`	Renders one-sided polygons. Renders the backface when set to off, meaning that you can't see though the back side of an object's polygons. This is helpful when trying to reduce polygon count.
`InvWorldMatrix`	Represents the inverse of the world matrix for an instance (concatenation of all the matrices of the parents of the instance). This property also can be used with `VETCamera`.

The following MTX code samples demonstrate how to use these properties.

```
<!--EXAMPLE 1: Materials. -->
<!--Here, display the name of the first material of the instance named
‘cube’. -->

<MTSJavaScript Func=”alert(MTSInstance.cube.Materials[0].name);”/>

<!--EXAMPLE 2: Geometry. -->
<!--Here, the scene’s geometry is not named. -->

<MTSInstance Name="myName" >
   <MTSGeometry Type="MTSCube" />    <!-- no name for the geometry -->
</MTSInstance>

<!--Here, the Geometry property sets the geometry for the instance,
“myName,” with the rens property (which changes the render style). -->

<MTSSetProperty Target="MTSInstance.myName.Geometry.rens" Value="1" />

<!--This property also enables you to set or copy the geometry from one
instance to another. Now, instances “inst1” and “inst2” will have same
geometry. -->

<MTSSetProperty Target="inst1.Geometry" Value="inst2.Geometry" />

<!--Here, “inst1” takes on the geometry named, “myName_Geometry_Name.”
-->

<MTSSetProperty Target="inst1.Geometry" Value="myName_Geometry_Name"/>

<!--EXAMPLE 3: RenderStyle -->
<!--Change RenderStyle with an action-->

<MTSAction Name="MTSInstanceRenderChanger">
   <MTSAssignProperty Target="MTSInstance.MTSSimple_2.RenderStyle"
    Value="13" />
   <MTSAssignProperty Target="MTSScene.nerd" Value="1" />
</MTSAction>

<!--EXAMPLE 4: BackFaceDirection -->
<!--Change face direction with an action-->

<MTSAction Name="MTSInstanceRenderChanger">
   <MTSAssignProperty
```

```
    Target="MTSInstance.MTSSimple_2.BackFaceDirection" Value="-1" />
   <MTSAssignProperty Target="MTSScene.nerd" Value="1" />
</MTSAction>

<!--EXAMPLE 5: BackFaceCulling -->
<!--Change culling with an action-->

<MTSAction Name="MTSInstanceRenderChanger">
   <MTSAssignProperty
    Target="MTSInstance.MTSSimple_2.BackFaceCulling" Value="1" />
   <MTSAssignProperty Target="MTSScene.nerd" Value="1" />
</MTSAction>

<!--EXAMPLE 6: InvMatrix -->
<!-- Here, we use an interactor to copy the inverted world matrix from
Simple_1 to the world matrix of Simple_2 effectively mirroring the
position of Simple_1. -->

<MTSInteractor>
   <Target Name="Simple_0"/>
   <MTSHandle Event="MouseLeftClick" Action="MTSCopyProperty"
    Src="Simple_0.InvWorldMatrix" Dest="Simple_1.WorldMatrix"/>
</MTSInteractor>
```

MTSMaterial Tags and Properties

`MTSMaterial` features three new tags and properties.

Tag and Property	Description
`DiffuseTexture`	Points to the diffuse texture of the material.
`EnvTexture`	Points to the lightmap texture of the material.
`BumpTexture`	Points to the bumpmap texture of the material.

The following MTX code samples demonstrate how to use these tags and properties.

```
<!--EXAMPLE 1: -->
<!--Set the diffuse texture, "myTexture," on the material named
"myMaterial" -->

<MTSSetProperty Target="MTSMaterial.myMaterial.DiffuseTexture"
Value="myTexture" />

<!--EXAMPLE 2: -->
<!--Set the diffuse texture, "myTexture," on the first material of the
instance named "myInstance" -->

<MTSSetProperty
Target="MTSInstance.myInstance.Materials[0].DiffuseTexture"
Value="myTexture" />
```

MTSCookie Tags and Properties

Just like browser cookies, `MTSCookie` allows you to save variables or properties between runs of Viewpoint Media Player, allowing content to share its cookie information with other content, while providing security controls to prevent unauthorized content from reading the cookies.

In Viewpoint Media Player 3.0.14, `MTSCookie` has been enhanced to include another tag and property, `Extend`, that gives `MTSCookie` more flexibility in establishing cookie time specifications.

The Extend Tag and Property

Tag and Property	Description
`Extend`	Determines whether a specified `MTSCookie` operates on an absolute or relative time schedule for expiration. This means that, depending on the value affixed to `Extend`, a Viewpoint cookie can be: • Absolutely set to the first time a user loads content. • Relatively set to the last time a user loads content.

Usage Notes: If setting `MTSCookie` to 10 seconds:

- When `Extend="0,"` no matter how many times a user reloads the content, `MTSCookie` expires 10 seconds after the first time the content was loaded.
- When `Extend="1,"` each time a user reloads the content, the 10 second clock resets. So, if a user reloads the content after nine seconds have elapsed since last loading it, `MTSCookie` expires 19 seconds after it was first set.

Sample MTX Code

The following sample MTX code demonstrates how to use `Extend`.

```
<!--Here, two cookies are referenced, each assigned a different
expiration time schedule. -->

<MTSCookie>
   <MTSCookieTime Category="CookieOne" Lifetime="5" Extend="0" />
   <MTSCookieTime Category="CookieTwo" Lifetime="5" Extend="1" />
</MTSCookie>
```

Flash Features

Viewpoint Media Player 3.0.14 enhances support of the existing Flash component for the following:

- "Macromedia® Flash™ Support" More Flash ActionScript functions, allowing more effective communication between Flash and Viewpoint Media Player..
  Flash is declared in MTX code via `<MTSTimeElem Type="SWFView">`.
- "Macromedia® Flash™ Support" Increased support of native SVG tags and new Viewpoint-enhanced SVG tags.
  SVG is declared in MTX code via `<MTSTimeElem Type="SWFView">`.

Macromedia® Flash™ Support

Flash technology is a commonly accepted method of displaying interactive graphics on the Internet. It is the technology of choice for displaying animated advertisement banners, product presentations, online computer games, etc. Flash is vector-based, like SVG, and provides a high-level of file compression and high, resolution-independent image quality.

Viewpoint Media Player 3.0.14 introduces a major upgrade in support for the majority of Flash 5.0 ActionScript and further extends rendering capabilities. For general information on Flash 5.0 ActionScript, see Flash documentation.

ActionScript Functionality in Viewpoint Media Player 3.0.14

Viewpoint Media Player ActionScript functionality has been enabled to support the following Flash features:

Full Support	Partial Support	No Support
`Array`	`Date`	`Selection`
`Boolean`	`Object`	`Sound`
`Math`		`XML`
`Mouse`		`XMLNode`
`MovieClip`		`XMLSocket`
`Number`		
`String`		
`Arguments`		
`Color`		
`Key`		

Notes for Partial Support:

- **Date** The following methods of `Date` are supported: `getDate(), getDay(), getFullYear(), getHours(), getMilliseconds(), getMinutes(), getMonth(), getSeconds(), getTime()`
- **Object** Viewpoint Media Player supports the `Object` object as a data container, but object-oriented programming (OOP) functionality, such as inheritance and method overriding, is not currently supported.

The FillBackground Property

The `FillBackground` property provides flexibility in displaying the background of a Flash file.

Tag and Property	Description
`FillBackground`	Displays the background color of a specified .swf file as authored.

The following MTX code sample demonstrates how to use this property:

```
<!--Loading a Flash animation with its background color intact.
Replacing the value in FillBackground=”1” with “0” makes the
background transparent. -->

<MTSTimeElem Name="myName" Type="SWFView" On="1" Path="myFile.swf"
FillBackground="1" >
    <Target Name="MTSTexture.somename" />
    <Target Name="MTSTexture.secondname" />
</MTSTimeElem>
```

Flash-Specific Action Parameters

New Flash-specific action parameters have been added to enable Viewpoint Media Player to send mouse events to Flash. For example, you can send a `MouseMove` event to a specific location even when this event has not ocurred in the scene.

Parameter	Description
`MouseEvent`	Specifies an event, either standard, such as `MouseUp`, or custom, to the Flash file being used in a Viewpoint scene.
`x`	Specifies the mouse coordinate along the x axis to the Flash file being used in a Viewpoint scene.
`y`	Specifies the mouse coordinate along the y axis to the Flash file being used in a Viewpoint scene.

The following sample MTX code demonstrates how to override the events, `MouseLeftDown` and `KeyDown:A`, and instead, sending the `MouseMove` event to Flash.

```
<!--Here, in the MTX code the action is specified. -->
```

```
<MTSTImeElem Type=”SWFView” Name=”myFlash” />

<!--Here, the ‘fake’ events are specified using the parameters. -->

<MTSInteractor>
    <MTSHandle Event="MouseLeftDown" Action="myFlash" x="6"
     y="4" MouseEvent="MouseMove" />
    <MTSHandle Event="KeyDown:A" Action="myFlash" x="10"
     y="10" MouseEvent="MouseMove"/>
</MTSInteractor>
```

SVG Advanced Support

The SVG format is emerging through the cooperative efforts of the World Wide Web Consortium (W3C) and its members. SVG is an industry-accepted, vector-graphics XML format. In terms of functionality, it is somewhat similar to Flash, but it is a text format that you need only Windows Notepad to write. Viewpoint has worked to ensure that Viewpoint Media Player 3.0.14 increases SVG compatibility.

> **Note:** For more information about SVG, go to http://www.w3.org/Graphics/SVG/Overview.htm.

SVG in Viewpoint Media Player

SVG is a language for describing two-dimensional graphics in XML. SVG allows graphic objects in Viewpoint content, such as basic shapes and text. Viewpoint-supported SVG features include nested transformations, alpha masks, template objects, and extensibility.

One of the main advantages of using SVG in a Viewpoint scene includes SVG's XML-based syntax. For example, to modify an annotation, an arrow, and a tooltip for content that is already published with Flash requires you to edit the original source (.fla) file. With SVG, it is a matter of writing a couple of lines into the XML file.

SVG Tags Supported by Viewpoint Media Player 3.0.14

Viewpoint Media Player 3.0.14 supports a large subset of the entire suite of SVG elements, including those described in the following table.

Supported Features	Enhanced Features

SVG Document and coordinate systems:
 	
`<svg>`
	Text:
	
	`<text>`
	
	Viewpoint-specific attributes:
	
	`<text wrap-width>`
	`<text text-align>`
	`<text line-spacing>`
	`<text text-encoding>`
	`<text text-syntax>`

Grouping: `<g>`	Integrating .swf files: `<use>`

Basic shapes:
 	
`<rect>`
`<circle>`
`<ellipse>`
`<line>`
	Building frames:
	
	`<frame>`

Path shape: `<path>`	Getting SVG element boundaries: `bounds`

Supported Features	Enhanced Features

Text
 	
`<text>`
<tspan>
	User events syntax for any shape, text, or group:
	
	`onlclick`
	`onmousedown`
	`onmouseup`
	`onmouseover`
	`onmousemove`
	`onmouseout`

Gradients:
 	
`<linearGradient>`
`<radialGradient>`
`<stop>`
	Animation using Viewpoint XML

Loading image file: `<image>`	
Masking: `<mask>`	

Animation:
 	
`<animateTransform>`
`<animateColor>`
`<animate>`	

Notes for Supported Features:

- These tags function within a Viewpoint-enabled scene exactly as they do in a normal SVG Document. For more information on how to use these tags, see SVG support.
- **Text** Release 3.0.14 supports the listed text elements. However, it does not support the `<tref>` the `<textPath>` subelements.
- **Grouping** This includes parent-to-child propagation (inheritence) for all group properties.
- **Styling** The following attributes can be applied to any shape, text, or group: `style, fill, fill-opacity, stroke, stroke-width, stroke-opacity, visibilty, font-family, font-size, font-style`
- **Animation** The `<animate>` element is supported only partially, animating only these attributes: `stroke-width, stroke-opacity`, and `fill-opacity`.
- **Naming** The `id` attribute can be applied to any element.

Notes for Enhanced Features:

- **Animations** Any SVG attribute can be animated via standard Viewpoint animation protocols (`<MTSTimeElem>`) in MTX code.
- **DOM Syntax** SVG allows the Viewpoint-enabled scene to access any of its attributes via the DOM syntax. The user can set or get the value of any given SVG attribute, and even animate it using the `MTSTimeElem` animation feature. The DOM support makes it possible to build interactive, “smart” SVG via JavaScript; basically, all SVG attributes can be read or modified from JavaScript.
  For example, the DOM path for an SVG attribute is:

```
<!--“mySVGanim” is the name of the MTSTimeElem containing the
SVG Document, “SVG” is a specified tag, “myShape” is the
identification of the SVG element (for more information, see the
id attribute), and “fill” is the SVG attribute name of the
element. -->

<MTSSetProperty Path=”mySVGanim.SVG.myShape.fill”
Value="rgb(255 0 0)"/>
```

Viewpoint-Enhanced SVG Functionality

The following tables describe how to use the release 3.0.14 Viewpoint-enhanced SVG tags. These tags must be declared within the <svg> open and close tags within a scene.

Text

Description	Defines a graphics element consisting of text.
Tag	<text/>
Attributes	• **wrap-width** — Forces text lines that exceed the specified width to wrap to a new line, forming a paragraph. • **line-spacing** — Defines spacing for wrapped text lines; and is defined in multiples of font-size. (default 1.2). • **text-align** — Aligns text within the wrap-width space according to its values: left (default), right, center, justify, or none. • **text-encoding** — Enables multilingual support via its default value, “utf8” or enables ascii mode if given the value, “ascii.” • **text-syntax** — Enables support for html-like text syntax when given the value, “html.” The other available value, “svg,” disables html-like text syntax. The HTML text has to comply with XML syntax (in other words, the X-HTML format).
Usage Dependencies	Must be introduced as a subelement of <svg>.

Sample Code	`<MTSTimeElem Type="SWFView" On="1" >`
	`  <svg width="512" height="512">`
	`  <text y="50" fill="rgb(75,195,24)"`
	`  wrap-width="48" line-spacing="1.3"`
	`  text-align="center" text-`
	`  ecoding=”utf8”>  Viewpoint`
	`  </text>`
	`  </svg>`
	`</MTSTimeElem>`

Sample Code 2 HTML Syntax	`<MTSTimeElem Type="SWFView" On="1"` `PreAnimator="1" Loop="1">` `  <svg width="512" height="512">` `  <text x="0" y="50" text-syntax="html"` `  fill="#111" font-size="24" font-` `  family="times">` `  Html style` `  <font color="#ff0000">text</font>` `  here:  <b>Bold</b><i>Italic</i>` `  Hi` `  </text>` `  </svg>` `</MTSTimeElem>`

Note: Viewpoint Media Player 3.0.14 supports all of the native attributes for the <text> tag .

Integrating .swf Files

Description	Integrates SVG with Flash by loading a .swf file and positioning it in the SVG Document or by instantiating a single Flash movie clip (symbol) from the .swf file, which is specified outside of the SVG Document in the `<MTSTimeElem Path=”myFlash.swf”>` declaration.
Tag	`<use/>`

Attributes	• **transform** — Enables positioning of a .swf file within an SVG Document via its attributes: scale, translate, rotate, matrix.
	• **xlink:href** — Links the .swf file or symbol with the SVG Document. For example:
	  • xlink:href="test.swf"
	  • xlink:href=”swf#myClip”

Usage Dependencies	Must be introduced as a subelement of `<svg>`.

Sample Code	`<!--Here, the .swf file, “bubble,” is loaded into the SVG Document. -->`
	
	`<MTSTimeElem Type="SWFView" Name="bubble" On="1" Path=”bubble.swf”>`
	`  <svg width="800"  height="600" >`
	
	`<!--Here, the movie clip, top, is instantiated at the specified x and y positions. -->`
	
	`  <use transform=”translate(100,120)” xlink:href="swf#top" />`
	
	`<!--Here, another .swf file is loaded at a different x and y positions. -->`
	
	`  <use transform=”translate(200,120)” xlink:href="anotherFlash.swf" />`
	`  </svg>`
	`</MTSTimeElem>`

Note: For advanced Flash developers, to author Flash files that can be used in a SVG Document as described by xlink:href=”swf#myClip,” the myClip movie clip must be an exported library symbol.

Building Frames

Description	Arranges the child elements, which can be tiled or stretched using the layout attribute.
Tag	`<frame/>`
Attributes	• **layout** — Defines the layout for each child element within the frame, either via its default value, "tile," or "stretch." • **tile-width** — Defines the width of each child element within the frame, overriding the default bounds width of the child. • **tile-height** — Defines the height of each child element within the frame, overriding the default bounds width of the child. • **frame-order** — Defines the rendering order of the child elements of the frame by its default value, "normal," or "reverse." • **frame-style** — Relates to the layout value, either applying the layout style according to its default value, "vertical" or "horizontal." • **frame-anchor** — Sets an anchor point for the entire frame, fixing the frame position relative to the fixed anchor position. Its default value is the top left corner of the frame.
Usage Dependencies	Must be introduced as a subelement of `<svg>`.

Sample Code	`<!--Here, the red and blue rectangles tile vertically within the frame. The circle stretches vertically over both rectangles. The rendering order of the shapes (“normal”) depends on their declaration order in the code. -->`
	
	`<MTSTimeElem Type="SWFView" On="1">`
	`<svg width="512" height="512">`
	`<frame frame-style="vertical"`
	`frame-order="normal">`
	`<rect layout="tile" width="100"`
	`height="100" fill="red"/>`
	`<rect layout="tile" width="100"`
	`height="100" fill="blue"/>`
	`<circle layout=”stretch” cx="50"`
	`cy=”50” r="50" fill="green"/>`
	`<frame-anchor/>`
	`</frame>`
	`</svg>`
	`</MTSTimeElem>`

Getting Graphical Element Boundaries

Description	When used with the `MTSGetProperty` function, returns the width, height, x, or y boundaries of any SVG shape, text, or group.
Attributes	`bounds`
Usage Dependencies	Read-only property for SVG elements.

Sample Code	`<!--Here, we introduce the SVG Document in MTX code. -->`
	
	`<MTSTimeElem Type="SWFView"  Name="bubble" On="1">`
	`  <svg width="800"  height="600" >`
	`  <text id="Msg" x="10"  y="33"`
	`  onclick="clickMsg" >`
	`  Hello`
	`  </text>`
	`  </svg>`
	`</MTSTimeElem>`
	
	`<!--Here, mouse events must be enabled as follows. -->`
	
	`<MTSInteractor>`
	`  <MTSHandle Event=”LeftMouseClick”`
	`  Action=”bubble” />`
	`</MTSInteractor>`
	
	`!--This function displays a dialog box containing the bounds for the text named Msg, which returns the width value of the “Hello” text. -->`
	
	`<MTSAction Name="clickMsg">`
	`  <MTSJavaScript Func="alert(bubble.SVG.Msg.bounds.width);” />`
	`</MTSAction>`

Handling Mouse Events

Description	Specifies a scene event based on mouse interaction with the targeted object. Additional parameters can be specified on the event.
Events	• **onclick** — Left or right mouse button is pressed and released on the shape, text, or group. In the MTX code, define whether the onclick event covers both left and right mouse buttons. • **onmousedown** — Left or right mouse button is pressed on the shape, text, or group. • **onmouseup** — Left or right mouse button is released on the shape, text, or group. • **onmouseover** — Mouse enters the shape, text, or group from outside the shape, text, or group. • **onmousemove** — Mouse moves within the shape, text, or group. • **onmouseout** — Mouse exits the shape, text, or group.
Usage Notes	The SVG-related mouse events can be assigned to Groups as well as single shapes. When a shape is clicked, it is checked to see if it has an `onmouse` attribute. If it does not, then the parent group is searched, and so on. Groups can handle mouse events, and shapes within the group take precedence, overriding the events. The following parameters are set by default: • **ID** — The shape or group that handles the mouse event. • **Object** — The actual object that handles the mouse event. Note that you can further access ANY property of that object. For example, `Object.fill` or `Object.width` (assuming the object has these attributes). • **Point** — The 2D point of the mouse event (returns the x and y coordinates where the mouse clicked or moved). Additional parameters can be set. For example: `onclick="actionName(param1='value', param2="value")"`

Sample Code

```
<!-- Here, when the user clicks on the circle
object, the onclick calls the MTSAction
JavaScript function below.-->

<MTSTimeElem Type=”SWFView” Name=”tx” On=”1”>
   <svg width="512" height="400"
      <circle cx="370" cy="100" r="10"
       fill="red"
       onclick="func(myVar="11")"/>
   </svg>
</MTSTimeElem>

<MTSInteractor>
   <MTSHandle Event=”LeftMouseClick”
    Action=”tx” />
</MTSInteractor>

<!--The onclick declaration in the SVG
Document calls this action from the MTX code.
-->

<MTSAction Name="func">
      MTSJavaScript Func="alert(myVar)" />
</MTSAction>
```

Video Features

With the Video component’s release, Viewpoint Media Player becomes the first graphics player to interactively composite video content with many other media types, including Flash and 3D animations, creating the highest-quality rich media content for the Internet.

Because the Video component can be deployed on a standard http server, it can be used in secure corporate environments or across complex networks. Now, any sized organization can manage and deploy interactive video content across its infrastructure.

Video Key Features

The Video component delivers the following advantages:

- The best audio and video quality in today's market
- Integrates with other Viewpoint-supported media types, such as 3D animations.
- Immediate video streaming or cached delivery
- Dynamic Stream Switching™ (stream switching), Viewpoint's proprietary technique for managing compressed video files
- Highest compression rates with minimal file degradation
- Maintains frame rates of up to 30 frames per second
- Supports most web browsers on Windows-based systems
- Effective Digital Rights Management and reuse of assets across multiple channels

Video Tags and Properties

The Video component introduces a new suite of Viewpoint XML tags, properties, and events enabling you to fully utilize the component's functionality. For more information on how to use the Video component, including step-by-step usage procedures, a description of Video’s

technical architecture, and a FAQ list, see related documentation on Viewpoint Developer Central.

Video Events

With the introduction of the Video component comes several new Video-specific events, listed here:

Event	Description
`DownloadStart`	Fired when the video file begins downloading.
`MTSLoadDone`	Fired when the video file is completely downloaded.
`MTSAnimEnd`	Fired when the animation ends.
`VideoReady`	Fired when the video file has downloaded enough to play.
`DownloadError`	Fired when download does not complete.
`VideoStarted`	Fired when video has begun (when lead switching is used, this event occurs twice.
`VideoComplete`	Fired when video has completed (when lead switching is used, this even occurs twice).
`VideoSwitched`	Fired when one video has switched to another.
`StateChanged`	Fired once you get this event, you can get the state.

TalkNow Features

With TalkNow, Viewpoint scenes can be enhanced with spoken content, thereby bringing content and sound together for a compelling presentation.Viewpoint Media Player 3.0.14 introduces an enhancement to its TalkNow component: the `Volume` tag and property.

The Volume Tag and Property

The `Volume` tag and property gives greater flexibility in applying sound to different TalkNow animators in a scene.

Tag and Property	Description
`Volume`	Sets the volume (default value is “100”) for a tageted TalkNow animator. The volume setting does not affect the user’s machine or the entire Viewpoint scene.

The following sample MTX code demonstrates how to use this tag and property:

```
<!--EXAMPLE 1: Simple animation:-->
```

```
<MTSTimeElem Name="playit" Type="VMPSpeech" Path="rainbow.mts" On="1"
Volume="90" />
```

Chapter 4: Release 3.0.13 New and Enhanced Features

This chapter summarizes the new and enhanced features introduced in Viewpoint Media Player release 3.0.13 according to the following component categories, including brief descriptions of all new tags and properties.

- "SceneComponent Features" — Includes time and random number generation features, a suite of `MTSGeometry` tags and properties, and more.
- "ZoomView Features" — Includes tags and properties that enable higher degree of authoring control over ZoomView animations, including tile removal.
- "MTSLensFlares Features" — Includes one new feature, a tag enabling pre-set lens flare effects to be applied to a scene.
- "Flash Features" — Includes one new feature, a tag that increases the rendering quality of Viewpoint-integrated Flash files.

SceneComponent Features

SceneComponent is the main component of Viewpoint Media Player. Among other functions, this component parses .mtx files, creates objects in scenes, evaluates animations, manages events, and prepares scenes to be rendered by the renderer.

Viewpoint Media Player 3.0.13 introduces several key new features and enhancements for existing features, including:

- "VETTimeInfo" — This new feature includes tags and properties that get local and universal time information.
- "VETRandomGenerator" — This new feature includes tags and properties that generate random strings based on range specifications.
- "MTSGeometry Tags and Properties" — Includes new tags and properties that enable access to geometry components, such as UVs and Normals.
- "MTSInteractor Tags and Properties" — Includes a new tag and property that enables you to define a hierarchy of interactors.
- "MTSSceneParms Tags and Properties" — Includes two new tags and properties for animator frame evaluation and mouse control in HTML.

VETTimeInfo

Viewpoint Media Player release 3.0.13 introduces the `VETTimeInfo` feature. This new feature allows you to request, and in one instance set (using `SecondsSinceStartup`), local and universal time (UTC) information.

Tag and Property	Description
`SecondsSinceStartup`	Gets and sets the number of seconds elapsed since a specified object was created according to local time.

Tag and Property	Description
`Year`	Gets (read-only) the current year according to local time. For example, if called during the year 2003 (according to local time), the property would return "2003".
`Month`	Gets (read-only) the current month according to local time. For example, if called during the month of October (according to local time), the property would return "10".
`DayOfWeek`	Gets (read-only) the current day of the week according to local time. For example, if called during on Monday (according to local time), the property would return "1".
`Day`	Gets (read-only) the current day of the month according to local time. For example, if called during on the fourth day of the month (according to local time), the property would return "4".
`Hour`	Gets (read-only) the current hour of the day according to local time. For example, if called at midnight (according to local time), the property would return "0".
`Minute`	Gets (read-only) the current minute of the hour according to local time. For example, if called at the half-hour mark of any hour (according to local time), the property would return "30".
`Second`	Gets (read-only) the current second of the minute according to local time. For example, if called at the quarter mark of any minute (according to local time), the property would return "15".

Function	Description
`SecondsSinceToday`	Gets (read-only) the current second of the day according to local time. For example, if called at the strike of 00:01, a day's first minute (according to local time), the property would return "60".

Function	Description
`UTCYear`	Gets (read-only) the current year according to universal time (UTC). For example, if called during the year 2002 (according to universal time), the property would return "2002".
`UTCMonth`	Gets (read-only) the current month according to universal time (UTC). For example, if called during the month of October (according to universal time), the property would return "10".
`UTCDayOfWeek`	Gets (read-only) the current day of the week according to universal time (UTC). For example, if called on Monday (according to universal time), the property would return "1".
`UTCDay`	Gets (read-only) the current day of the month according to universal time (UTC). For example, if called during on the fourth day of the month (according to universal time), the property would return "4".
`UTCHour`	Gets (read-only) the current hour of the day according to universal time (UTC). For example, if called at midnight (according to universal time), the property would return "0".
`UTCMinute`	Gets (read-only) the current minute of the hour according to universal time (UTC). For example, if called at the half-hour mark of any hour (according to universal time), the property would return "30".
`UTCSecond`	Gets (read-only) the current second of the minute according to universal time (UTC). For example, if called at the quarter mark of any minute (according to universal time), the property would return "15".
`UTCSecondsSinceToday`	Gets (read-only) the current second of the day according to universal time (UTC). For example, if called at the strike of 00:01, a day's first minute (according to universal time), the property would return "60".

Function	Description
`UTCDifference`	Gets (read-only) the difference between the local time and universal time (UTC) in seconds. For example, if called at any minute of any day EST, the property would return "18000" because there are 5 hours (5 hours at 60 minutes/hour = 18000 seconds) difference between UTC and EST. The same calculation process works for all time zones, yielding results specific to only that time zone.

The following MTX code samples demonstrate how to use these features:

```
<!--EXAMPLE 1: GENERIC -->
<!-- This sample can be used for all of the VETTimeInfo properties by
plugging in any of the properties in the “InsertFunction” value where
TimeInfo::InsertFunction.-->

<VETTimeInfo Name="TimeInfo" />
    <MTSInstance>
    <MTSGeometry Name="MESH_0" />
    <OnClick Action="MTSJavaScript"
     Target="alert(TimeInfo::InsertFunction)" />
</MTSInstance>

<!--EXAMPLE 2: SecondsSinceStartup -->
<!--To set the value to "0" seconds  -->
<!--Any set value becomes the elapsed time from an object's creation to
the current time. To reset the counter that records how many seconds
elapse between the content's creation and the present time, set this
property's value to "0". -->

<VETTimeInfo Name="TimeInfo" />
    <MTSInstance>
    <MTSGeometry Name="MESH_1" />
    <MTSHandle Event="MouseLeftDown" Action="MTSSetProperty"
     Target="TimeInfo::SecondsSinceStartup" Value="0" />
</MTSInstance>
```

Usage Notes:

- All `VETTimeInfo` functions are read-only *with the exception of* `SecondsSinceStartup`.
- **`SecondsSinceStartup`**: An object's creation time is equivalent to when the content was parsed in the .mtx file where it resides (according to local time).
- **`Month`** and **`DayOfWeek`**: To convert the number value to the actual month (such as October), use JavaScript to trigger a function that will convert the number to a string. For example:

```
<MTSHandle Event="MouseLeftDown" Action="MTSJavaScript"
Target="setDay(TimeInfo::Month)" />
<!--Declare this in MTX code and call the function setDay and by
passing in the value in the parenthesis, it can be used with a
case statement or if/else statement to make the conversion. -->
```

VETRandomGenerator

Viewpoint Media Player release 3.0.13 introduces the `VETRandomGenerator` feature. This feature allows for random number generation according to specified ranges.

For information on `VETRandomGenerator` features introduced in release 3.0.14, see "VETRandomGenerator Tags and Properties".

Tag and Property	Description
`Range`	Specifies the low- and high-number threshold for random number generation. When the values for this tag are defined, `Range="0 1000"`, the random number generated by `VETRandomGenerator` falls between 0 and 1000.
`Random`	Gets (read-only) the random number generated by `VETRandomGenerator` according to the pre-specified high- and low-number thresholds for the number. This property can be used with GetProperty or alerts in JavaScript to get a random number.
`RangeMax`	Specifies the high-number threshold for random number generation. This tag is the same as `Range.y`.
`RangeMin`	Specifies the low-number threshold for random number generation. This tag is the same as `Range.x`.
`Gap`	Specifies the minimal numerical difference between random numbers generated via VETRandomGenerator. For example, when the value for this tag is defined, `Gap="5"`, the random number generated represents only a value separated by 5 real numbers, such as 5, 10, and 15, depending on the minimum and maximum range thresholds.

The following MTX code examples demonstrate how to use these tags and properties:

```
<!--EXAMPLE 1: Range -->

<VETRandomGenerator Name="myrandom" Range="0 2.9"/>
<MTSAction Name="rs2">
    <MTSAssignProperty Target="theStuff::rwnd" Value="1"/>
    <MTSAssignProperty Target="theStuff::time"
     Value="myrandom::Random"/>
</MTSAction>
<MTSAction Name="random_stuff">
    <MTSSetProperty Target="MTSScene::newProp" Value="0"/>

<!--EXAMPLE 2: RangeMin and RangeMax -->
```

```
<MTS_Scene Version="313" />
   <VETRandomGenerator Name="myrandom" RangeMin="1" RangeMax="1000" />
   <MTSInstance Name="sphere_1txt">
      <MTSHandle Event="MouseLeftDown" Action="MTSJavaScript"
       Target="alert(myrandom::Random)" />
</MTSInstance>
```

MTSGeometry Tags and Properties

MTSGeometry is a subelement that contains settings for the polygons that make up an object. Viewpoint Media Player 3.0.13 introduces a host of new `MTSGeometry` tags and properties that allow precision UV, vertex, normal, and indices controls.

Tag and Property	Description
`Vertices`	Defines the array of the individual vertices in a scene's geometry.
`Normals`	Defines the array of the different normals used in a scene's geometry. For example, this property can be used to animate an independent normal value.
`UVs`	Defines the array of the different UVs used in a scene's geometry. Can be used to access an individual UV set in a scene's geometry.
`VertexIndices`	Specifies the array of vertex indices used to create triangles in a scene's geometry. The maximum index value should be less than the number of vertices.
`UVIndices`	Specifies which UV to use for every corner of every triangle of a scene's geometry.
`NormalIndices`	Specifies which normal to use for every corner of every triangle of a scene's geometry.
`MaterialIndices`	Specifies the array of material indices to be used for each face of a scene's geometry. This array can be empty. If it is not empty, it should be the same size as the `VertexIndices` array divided by three. (Each index relates to a face, not a vertex.)
`SmoothingGroups`	Specifies which triangle belongs to a certain smoothing group.
`MultiUV`	Enables the use of a second set of UVs for lightmap rendering instead of normals in a scene's geometry.

The following MTX code samples demonstrate how to use these properties:

```
<!--EXAMPLE 1: Vertex and VertexIndices -->
```

```
<!--Create vertices in a scene-->
<MTSInstance Name="my_instance" >
    <MTSGeometry Name="my_GEOM" >
        <Vertices>[1 1 0][1 0 0][-1 0 0]</Vertices>
        <VertexIndices>[0][1][2]</VertexIndices>
    </MTSGeometry>
</MTSInstance>

<!--EXAMPLE 2: UVs and UVIndices -->
<!--Create UVs in a scene-->
<MTSInstance Name="my_instance" >
    <MTSGeometry Name="my_GEOM" >
        <UVs>[1 1 0][1 0 0][-1 0 0]</UVs>
        <UVIndices>[0][1][2]</UVIndices>
    </MTSGeometry>
    <MTSMaterial Name="my_mat" ID="0" >
        <MTSTextureMap Type="Diffuse" Name="my_tex"/>
    </MTSMaterial>
</MTSInstance>

<!--EXAMPLE 3: Normals and NormalIndices -->
<MTSInstance Name="my_instance" >
    <MTSGeometry Name="my_GEOM" >
        <Normals>[1 1 0][1 0 0][-1 0 0]</Normals>
        <NormalIndices>[0][1][2]</NormalIndices>
    </MTSGeometry>
</MTSInstance>

<!--EXAMPLE 4: MaterialIndices -->
<!--Associate material 0 to face 0,1 and 2. Associate material 1 to
face 3-->
<MTSInstance Name="my_instance" >
    <MTSGeometry Name="my_GEOM" >
        <MaterialIndices>[0][0][0][1]</MaterialIndices>
    </MTSGeometry>
</MTSInstance>

<!--EXAMPLE 5: SmoothingGroups -->
<!--Create smoothing groups in a scene -->
<MTSInstance Name="my_inst" >
    <MTSGeometry Name="my_GEOM" >
        <Vertices>[1 1 0][1 0 0][-1 0 0][-1 0 0][-1 1 1][1 1
         0]</Vertices>
        <VertexIndices>[0][1][2][2][4][0]</VertexIndices>
        <SmoothingGroups> [1][1] </SmoothingGroups>
    </MTSGeometry>
</MTSInstance>

<!--EXAMPLE 6: MultiUV -->
<MTSInstance Name="my_instance">
    <MTSGeometry Name="my_geometry" MultiUV="1" />
</MTSInstance>
```

Usage Notes:

- **`Vertices`**: Can be used to access an individual vertex in a scene's geometry. For example, this tag can be used to:
  - Animate only one vertex without using a morph animation;
  - Entirely define the geometry in the .mtx file without an .mts file.
- **`Normals`**: When modifying a geometry's vertex coordinate, Viewpoint Media Player automatically recomputes the `Normal` for this geometry. Therefore, in this case, any Normal change that you have specified is invalidated. For this reason, when animating both `Vertex` coordinates and `Normals` values, place the `Vertex` animation before the `Normals` animation in the MTX code.
- **`UVs and UVIndices`**: Don't specify two elements of the array with the same values. Instead, specify the same index in `UVIndices`.
- **`VertexIndices`**: To define a geometry, define triangles one by one, specifying which vertex to use for each corner of the triangles. The number of triangles in a geometry is equal to the number of elements in `VertexIndices` divided by the number three.
- **`SmoothingGroup:`** The normals of triangles belonging to compatible smoothing groups are smoothed together. Two smoothing groups are compatible if the AND bitwise operation of their number is not null. For example, groups 1 and 2 are not compatible whereas 1 and 3 are compatible, 2 and 3 are compatible.
  This array can be empty. If it is not empty, its size should be the number of triangles in the geometry. Therefore, the number of elements of `VertexIndices` divided by three.
- **`MultiUV`**: When enabled, this property effectively 'pins' a lightmap to an object surface just like a texture, giving the surface the look of a 2-layer texture. Use this property only to enable or disable the use of the geometry's UVs, which should be present in an .mts file. If the UVs are not present in your .mts file, this property cannot affect a scene. If the UVs are present in the .mts file, but you define `MultiUV="0"`, the UVs are ignored and the lightmap is rendered as usual.

MTSInteractor Tags and Properties

Viewpoint Media Player 3.0.13 introduces a new `MTSInteractor` tag and property, `ActivationState`, making it possible to automatically activate or deactivate a child interactor based on the state of the parent interactor.

Tag and Property	Description
`ActivationState`	Automatically activates or deactivates the child interactor when the state of its parent is equal to its `ActivationState` setting, allowing you to define a hierarchy of interactors.

The following MTX code sample demonstrates how to use this tag and property:

```
<!--In the following code, the child interactor will not become active
until state "second" (the end state of the parent interactor). -->
<!--Avoid naming an activation state with lone real numbers or
integers. For example, do not use <MTSInteractor ActivationState="4">,
but instead use, <MTSInteractor ActivationState="four">. -->

<MTSInteractor>
   <Target Name="MTSInstance.CubeInstance" />
   <MTSHandle StartState="First" EndState="Second"
    Action="MyActionScript"  Event="MouseRightClick" />

   <MTSInteractor ActivationState="Second">
   <MTSHandle Action="MTSJavaScript" Func="alert('coucou');"
    Event="MouseLeftClick" />
   </MTSInteractor>
</MTSInteractor>

<!--Of course, the child interactor could have become active at the
same time as the parent, as in the following. -->

<MTSInteractor>
   <Target Name="MTSInstance.CubeInstance" />
   <MTSHandle StartState="First" EndState="Second"
    Action="MyActionScript"  Event="MouseRightClick" />

   <MTSInteractor ActivationState="First">
   <MTSHandle Action="MTSJavaScript" Func="alert('coucou');"
    Event="MouseLeftClick" />
   </MTSInteractor>
</MTSInteractor>
```

MTSSceneParms Tags and Properties

Viewpoint Media Player 3.0.13 introduces two new `MTSSceneParms` tags and properties, `FPS` and `PassClick`, giving you significantly improved controls over mouse functionality.

Tag and Property	Description
`FPS`	Sets how many frames per second (FPS) play in a scene (how often animators are evaluated). Useful for an extremely slow animation (saves CPU usage), or an extremely fast animation (if it is very simple and is running on a very fast machine).
`PassClick`	Sets whether or not mouse clicks pass through Viewpoint Media Player to HTML code. For example, this property allows a link that is behind the Viewpoint Media Player window to be clickable.

The following MTX code samples demonstrate how to use each of these tags and properties:

```
<!--PassClick="1" allows the user to click through the Viewpoint Media
Player window, but not through the Viewpoint-enabled object. If
PassClick="2", the user can click through the Viewpoint Media Player
window and through Viewpoint-enabled objects. -->

<!--The FPS setting means the scene is evaluated every 1/100 seconds.
This property should not affect your scene's download speed; however,
```

```
interaction with the scene (such as dragging the mouse) can cause
content anomalies. -->

<MTSSceneParms RenderMode="LightTexture" PassClick="1" FPS=”100” />
```

Usage Notes:

- **`PassClick`**: Setting this property to anything but “0” affects camera navigation:
  - When `<PassClick="1">`, the user must click on an object in the scene to get normal camera navigation capabilities. If a user clicks the scene without clicking on a geometry, the user cannot move the camera.
  - When `<PassClick="2">`, the user can click through the scene regardless of the scene's objects, but has no control over camera navigation.

ZoomView Features

The ZoomView component supports Viewpoint ZoomView technology, which allows you to publish high-resolution images to the Internet. `MTSTimeElem` is the general Viewpoint animation element, and the `Type` attribute for ZoomView is `Type="ZoomView"`.

Tag and Property	Description
`KAlp`	Maintains an alpha channel in a ZoomView image target when the image is targeted to a texture. In the ZoomView animator, `KAlp="1"` makes the image take on the alpha that is applied to a hot spot texture.
`Blit`	Returns (read-only) the last size of the ZoomView target, such as a backbuffer or hot spot.
`Norm`	Sets the ZoomView coordinate type to enable you to use the ZoomView properties `zi_x`, `zi_y`, `iz_x`, and `iz_y`.
`zi_x`	Converts ZoomView coordinates, which represent the image's original (pre-ZoomView) pixel dimensions, to the Viewpoint Media Player window's x coordinate.
`zi_y`	Converts ZoomView coordinates, which represent the image's original (pre-ZoomView) pixel dimensions, to the Viewpoint Media Player window's y coordinate.
`iz_x`	Converts the Viewpoint Media Player screen position to the ZoomView x coordinate, which represents the image's original (pre-ZoomView) pixel dimensions.

Tag and Property	Description
iz_y	Converts the Viewpoint Media Player screen position to the ZoomView y coordinate, which represents the image's original (pre-ZoomView) pixel dimensions.

Tag and Property	Description
Clip	Sets an additional clipping rectangle, which directs ZoomView to re-render only in the specified coordinates.
Colo	Sets the color of the rectangle that is drawn around newly downloaded ZoomView tiles in RGB values.
RmvT	Specifies a list of ZoomView tiles to be left out of a scene.
OveZ	Sets the zoom ratio that allows linear and more natural zoom effects.
RmvL	Specifies a list of tile levels to be left out of a ZoomView scene.
AddT	Adds custom tiles into the ZoomView tile resource folder.

The following MTX code samples demonstrate how to use these tags and properties:

```
<!--EXAMPLE 1: Used as tags -->

<MTSTimeElem Name="mySmoothZoom" Type="MTSZoomView" On="1"
PreAnimator="1" Zfit="2" ImageSize="1280 1024" TileSize="128"
BaseName="my_zv_images/my_zv.mzv" RmvL="3" RmvT="0 1 2 3 4 5 6 7 8 9
10" Colo="255 0 0" Clip="0 0 200 250" Norm=”0” OveZ=”0”
AddT="my_zv_images/my_zv1B.mzv">
</MTSTimeElem>

<!--EXAMPLE 2: changing iz_y, iz_x, zi_y, and zi_x-->

<!--Setting these properties via an animation, substituting “Insert”
with the property name of choice. Remember, to set or get the zi_x or
zi_y values, set Norm="1." To get or set the iz_x and iz_y values, set
Norm="0.” The Norm property enables you to change between the two types
of coordinates. -->

<MTSTimeElem Type="Keyframe" Name="Insert_anim" On="0">
   <Target Name="mySmoothZoom" Property="Insert" Timeline="T1"/>
   <Time> 0 1 </Time>
   <Timeline Name="T1" Type="1D">
   * [1]
   </Timeline>
</MTSTimeElem>

<!--EXAMPLE 3: Getting the Blit value with a JavaScript function, which
returns the last size of the ZoomView target. -->

function get_Blit(){
vmp.GetProperty('MTSTimeElem.mySmoothZoom', 'Blit', 'mts_pnt2d');
}
```

Usage Notes:

- **`Norm`**: `Norm` and the properties, `iz_x`, `iz_y`, relate to the window or screen coordinate system in which the center of the screen corresponds to x and y values of 0.5 and 0.5. `Norm` and the properties, `zi_x` and `zi_y`, related to the original pixel dimensions (values between 0 and 1) of the ZoomView image before it was processed into ZoomView ("normalized" mode).
- **`RmvT` and `RmvL`**: Use `RmvT` to specify which tiles do not load in a scene. When setting this property, the specified values (tile numbers) are ignored by the animator and therefore never load. This property does not remove tiles from your actual image folder, just from the scene.
  Use `RmvL` to override the ZoomView tile database by specifying a list of tile levels to be removed from it. When setting this property, the specified values (tile levels) are ignored by the animator and therefore never load.
- **`OveZ`**: Use this property instead of `Zoom` to create more natural zoom animations (`OveZ = 1/Zoom`). Instead of getting the effect of zooming in and then panning over to a specific spot, `OveZ` provides a smooth zoom and pan at the same time.
- **`AddT`**: For example, to download only one tile for the first zoom position, use the `RmvL` property to erase tile level 3 and add a single tile to replace it.
- **`KAlp`**: This tag is used to apply alpha channels to ZoomView images. This is done by targeting the ZoomView image to a texture hot spot. When making the texture hot spot, make a simple white texture (which compresses best) and apply it to the hot spot. Then load an alpha map and target it to the texture that is applied to the hot spot, thereby applying the alpha on the hot spot. Note that the alpha functions until the ZoomView image loads in front of the alpha and texture.

MTSLensFlares Features

The MTSLensFlares component was introduced in Viewpoint Media Player 3.0.11, enabling innovative scene lighting effects.

Viewpoint Media Player 3.0.13 introduces the `defa` tag and property to compliment the existing `MTSLensFlares` feature suite

Tag and Property	Description
`defa`	Sets `MTSLensFlares` values to create a wide array of pre-set effects. Choose from 25 different pre-set effects.

The following MTX code sample demonstrates how to use this tag and property:

```
<!--defa as a tag:-->

<MTSTimeElem Name="myLensFlare1" Type="MTSLensFlares" defa="0" >
    <RefObject Name="Simple_0" />
</MTSTimeElem>
```

Note: In the `MTSLensFlares` builder section of the Viewpoint SceneBuilder authoring tool, there is a button that, when selected, displays list of pre-set lens flares effects. Choose one of these options instead of creating your own lens flare effect, or use one of these as a starting point to build one that suits your needs.

Flash Features

Viewpoint Media Player 3.0.13 introduces one new Flash-specific tag and property, `Quty`, that renders Flash movies using bilinear interpolation.

Tag and Property	Description
`Quty`	Renders Flash movies using bilinear interpolation. When `Quty="1"` (its default value), the Flash file displays smoother and less pixelated.

The following MTX sample code demonstrates how to use this tag and property:

```
<MTSTimeElem Type="SWFView" Name="flash1" On="1" Path="fire.swf"
Init="1" Quty="0"/>
```

Appendix A: Help, Resources, and Feedback

Viewpoint Developer Central: A Complete Resource

Viewpoint Developer Central is a complete resource for Viewpoint content developers. At this website, you can access Viewpoint applications, user guides, downloadable example files, support, production tips, and techniques – to name just a few of the offerings there.

Access Viewpoint Developer Central to:

- Get Assistance — For questions about using Viewpoint, click **Forums** under **Support** in the left navigation bar.
- Get Examples — Click **Examples & Tips** in the left navigation bar.
- Subscribe to the Viewpoint Developer Newsletter — Learn new production tips and techniques for creating 3D and rich media content for the web with Viewpoint. Click **Newsletter** in the left navigation bar.
- Give Feedback About Viewpoint Applications — Viewpoint Corporation values your feedback. Direct your comments and suggestions to the Viewpoint Forums.

You can also visit the Viewpoint Corporation website for company news, links to websites featuring Viewpoint content, and more.

Download Viewpoint Applications, Guides, and Examples

Viewpoint Developer Central is continually updated with the latest versions of applications, user guides, and examples. Find links to the following in the left navigation bar.

Viewpoint Applications

You can download Viewpoint applications free of charge. The following applications are available for download:

- Viewpoint Media Player — The web browser plug-in necessary to view Viewpoint content with Netscape Navigator or Internet Explorer.
- Viewpoint Scene Builder — An essential application for assembling a scene and publishing it in .html/.mtx/.mts format.
- Viewpoint Media Publisher — An application enabling you to quickly create Viewpoint scenes from Viewpoint media files (.mtx/.mtz) by embedding them in web (html) pages or running transformations on .mtx (xml) files through built-in XSLT support.
- Viewpoint Stream Tuning Studio — An application for reducing .mts file sizes, enabling optimized 3D scenes rendered on a web page to stream quickly and retain visual integrity.
- Viewpoint Control Panel — A utility for checking, installing, and removing individual Viewpoint Media Player components.

User Guides

For more information on the Viewpoint Platform, go to Viewpoint Developer Central and click on the **Reference** tab. From here you can access Viewpoint documentation, such as the *Viewpoint Rich Media Authoring Guide*, and the *Viewpoint XML Reference Database*.

Appendix B: Viewpoint Content Creation Updates

Viewpoint Content Checklist

Viewpoint 3.0.15 content has certain features and authoring requirements for display on the Windows platform:

- **Install the latest version of Viewpoint Media Player (version 3.0.15).**

  To ensure new Viewpoint content displays as expected, install the latest version of Viewpoint Media Player from Viewpoint Developer Central: http://developer.viewpoint.com/dc/index.shtml.

- **Use the latest version of MTS3Interface.js (currently version 4.0.5.4).**

  To ensure new Viewpoint content uses the correct version of MTS3Interface, you can publish it with the newest release of Viewpoint Media Publisher. Viewpoint Media Publisher contains the version of MTS3Interface that supports content viewed with Viewpoint Media Player 3.0.15 for Windows.

  To find out more about using MTS3Interface, review the *Viewpoint Rich Media Authoring Guide* from the **Reference** tab of Viewpoint Developer Central.

- **Correct DIV and Layer tags for Netscape 7.x.**

  This involves adding sniffer script available on web developer resource sites to detect the web browser type and version.

  Important: Do not create sniffer scripts that refer to variables (such as isIE or isNN) set in MTS3Interface.js, as this JavaScript file is modified periodically to accommodate new functionality, and variables contained in it may change.

- **Update required settings for your web page's Viewpoint content.**

  The type of rich media content you include in your web page dictates the components you must specify in your constructor call. See the Viewpoint Experience Technology Deployment Guide for information on how to refer to these requirements for your content.

- **Create content to avoid the opening of multiple browser windows containing Viewpoint content.**

  For best performance, only one browser window containing Viewpoint content should be open at a time.

Updating Existing Web Pages with MTS3Interface

MTS3Interface version 4.0.5.4 or higher is required for content you want to view with Viewpoint Media Player 3.0.15. 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 MTS3Interface dynamically builds an object embed and assigns a variable to it. It also facilitates interaction between Viewpoint Media Player and the web page, bi-directionally, across a variety of browsers and operating systems.

If you are already using MTS3Interface, simply replace the old file. If you are using an object embed style that does not refer to MTS3Interface, you need to update your HTML. Simply replace the old object/embed with a Viewpoint Media Player constructor line, and refer to the MTS3Interface in the head of your document. For step-by-step instructions, refer to the *Viewpoint Rich Media Authoring Guide*.

Example for the New Naming Convention

To change JavaScript calls between the HTML page and Viewpoint Media Player to the new naming convention:

1. With the name of your object variable, "vmp", by default:

   Change: `onclick="triggeranimation('anim1')`

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

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

Using One MTS3Interface.js File for a Website

Using one MTS3Interface.js file for a website allows you to easily take advantage of updates and improvements to this file's support for Viewpoint scenes. You need only to replace one file to update.

This also benefits users viewing your Viewpoint content: The MTS3Interface.js file is cached after a user views the first Viewpoint scene on your site, thereby speeding up all other pages viewed.

Verifying the MTS3Interface Version in Existing Content

When experiencing problems with Viewpoint 3.0.14 content, check which version of MTS3Interface the content uses. This method is applicable if the content:

- Scripts in/out of the Viewpoint scene with JavaScript
- Uses layers
- Runs in Netscape 7 (or on a Mac or Compuserve)

To check what MTS3Interface a piece of content is using:

1. Type the following command into the browser address line while the web page displays:

   `javascript:alert(VET_IfVer)`

2. Look for the diplayed value of the VET_IfVer variable, which is in all of the MTS3Interface files from version 4 onwards.

   If a lower number than 04.00.05.04 displays, this could be the problem for why the content is not performing as expected.

If the VET_IfVer command gives an error and an error display box appears:

1. Click “No” in the error message box that displays asking if you want to debug.
2. Then, use the command:

```
javascript:alert(_ver)
```

Note: If this process still does not get the version of MTS3Interface used by Viewpoint content, one of two possibilities explains why: 1) you are using a very outdated MTS3Interface version; or 2) you are not using the MTS3Interface at all.