



Adobe® Experience Cloud

Adobe Viewers Reference Guide

Contents

- System requirements.....7**
- Installing the viewers on the same server.....9**
- Compatibility notes.....10**
- Viewers release notes (5.10.1).....12**
 - Release notes archive.....13
 - Scene7 Viewers 5.8.2 Release Notes.....13
 - Scene7 Viewers 5.5.2 Release Notes.....15
 - Scene7 Viewers 5.4.2 Release Notes.....16
 - Scene7 Viewers 5.2.3 Release Notes19
 - Scene7 Viewers 5.2.2 Release Notes.....21
 - Scene7 Viewers 5.1.1 Release Notes.....23
 - Scene7 Viewers 5.0.1 Release Notes.....25
 - Scene7 Viewers 4.9.2 Release Notes.....27
- Viewers for AEM Assets and Scene7.....30**
 - HTML5 Basic Zoom Viewer.....30
 - Command reference – Configuration attributes.....36
 - Javascript API reference for Basic Zoom Viewer.....43
 - Event callbacks.....49
 - Customizing Basic Zoom Viewer.....49
 - Support for Adobe Analytics tracking.....61
 - Localization of user interface elements.....62
 - Full Screen Support.....63
 - Viewer SDK namespace.....63
 - eCatalog.....64
 - Command reference – Configuration attributes.....71
 - Javascript API reference for eCatalog Viewer.....93

Event callbacks.....	101
Customizing eCatalog Viewer.....	101
Support for Adobe Analytics tracking.....	199
Localization of user interface elements.....	200
Image map support.....	205
Managing page labels.....	206
Full screen support.....	207
Print feature.....	207
Download.....	207
Favorites feature.....	207
Viewer SDK namespace.....	208
eCatalog Search.....	208
Command reference – Configuration attributes.....	215
Javascript API reference for eCatalog Search Viewer.....	240
Event callbacks.....	248
Customizing eCatalog Search Viewer.....	248
Support for Adobe Analytics tracking.....	352
Localization of user interface elements.....	353
Image map support.....	359
Managing page labels.....	360
Print feature.....	361
Full screen support.....	361
Download.....	361
Favorites feature.....	361
Viewer SDK namespace.....	362
Flyout.....	362
Command reference – Configuration attributes.....	369
JavaScript API reference for Flyout Viewer.....	379
Event callbacks.....	385
Customizing Flyout Viewer.....	386
Support for Adobe Analytics tracking.....	396
Localization of user interface elements.....	397
Viewer SDK namespace.....	398
Inline Zoom.....	399

Command reference – Configuration attributes.....	407
JavaScript API reference for Inline Zoom Viewer.....	415
Event callbacks.....	421
Customizing Inline Zoom Viewer.....	422
Support for Adobe Analytics tracking.....	431
Localization of user interface elements.....	432
Viewer SDK namespace.....	433
Mixed Media.....	434
Command reference – Configuration attributes.....	442
Javascript API reference for Mixed Media Viewer.....	469
Event callbacks.....	476
Customizing Mixed Media Viewer.....	476
Support for Adobe Analytics tracking.....	512
HTTPS video delivery.....	514
Localization of user interface elements.....	514
Full Screen Support.....	516
Viewer SDK namespace.....	516
HTML5 Spin Viewer.....	517
Command reference – Configuration attributes.....	524
Javascript API reference for Spin Viewer.....	532
Event callbacks.....	538
Customizing Spin Viewer	538
Support for Adobe Analytics tracking.....	553
Localization of user interface elements.....	554
Full screen support.....	555
Viewer SDK namespace.....	555
Video.....	556
Command reference – Configuration attributes.....	563
Command reference – URL.....	574
JavaScript API reference for Video Viewer.....	576
Event callbacks.....	584
Customizing Video Viewer.....	584
Support for Adobe Analytics tracking.....	647
HTTP video delivery.....	648

Localization of user interface elements.....	649
Full screen support.....	652
External video support.....	653
Viewer SDK namespace.....	653
Zoom.....	654
Command reference – Configuration attributes.....	661
Javascript API reference for Zoom Viewer.....	672
Event callbacks.....	678
Customizing Zoom Viewer.....	679
Support for Adobe Analytics tracking.....	695
Localization of user interface elements.....	696
Full Screen Support.....	697
Viewer SDK namespace.....	697

Viewers for AEM Assets only699

Interactive Images.....	699
Command reference – Configuration attributes.....	705
Command reference – URL.....	706
JavaScript API reference for Interactive Image Viewer.....	707
Event callbacks.....	712
Customizing Interactive Image Viewer.....	713
Support for analytics tracking.....	719
Hotspot support.....	720
Preload image.....	720
Viewer SDK namespace.....	720
Interactive Video.....	721
Command reference – Configuration attributes.....	728
Command reference – URL.....	745
JavaScript API reference for Interactive Image Viewer.....	747
Event callbacks.....	754
Customizing Interactive Video Viewer.....	755
Support for Adobe Analytics tracking.....	798
HTTPS video delivery.....	799
Interactive data support.....	801

Localization of user interface elements.....	801
Full screen support.....	803
Viewer SDK namespace.....	803
Carousel.....	804
Command reference – Configuration attributes.....	810
Command reference – URL.....	815
JavaScript API reference for Carousel Viewer.....	816
Event callbacks.....	822
Customizing Carousel Viewer.....	823
Support for Adobe Analytics tracking.....	834
Localization of user interface elements.....	834
Hotspot and Image maps support.....	835
Preload image.....	836
Viewer SDK namespace.....	836

Command reference common to all viewers – Configuration attributes.....837

stageSize.....	837
style.....	837
title.....	838

Command reference common to all viewers – URL.....839

asset.....	839
caption.....	842
config.....	843
config2.....	845
contentUrl.....	846
initialFrame.....	846
serverUrl.....	847
videoServerUrl.....	847

Keyboard accessibility and navigation.....849

Viewer SDK Tutorial.....851

System requirements

System requirements for HTML5 viewers.

Server hardware and software

- Scene7 Image Serving 6.5.3 or later.
- HTML5 Viewers require SDK JavaScript server-side libraries 3.5. or later.
- "Email a Friend" social features require s7ondemand 4.8.1 or later.
- eCatalog Viewer – Info panel support requires info server 2.1.5 or later.
- Search feature components require s7search 2.2.0 or later.

\



Note:

Viewers system requirements

Client browser minimum requirements for component viewers

- Color monitor and video card that supports 16-bit High Color at 1024x768 resolution or higher.
- Microsoft® Windows® 7 or later; Mac OS X 10.9 or later. 512 MB.
- Firefox 57, Safari 10 (Mac OS only), Chrome 63 (or latest version; Chrome updates automatically to the latest version), Internet Explorer 11.0 or later, Microsoft Edge.
- iOS6 or later.
- Certified on iPhone 3GS or later, and iPad 2 or later (Safari and Chrome browsers only).
- Android OS 4.4 or later.
- BlackBerry 10 or later; native browsers only. Video playback is only supported at this time.
- Internet Explorer on mobile devices is not supported at this time.
- Panoramic Viewer supported on iOS 10 or later, Firefox 57, Chrome 63, Safari 11 or later, Internet Explorer 11 or later, Microsoft Edge, Android 4.4 or later (phone devices only).
- Zoom Vertical viewer is supported on iOS 10 or later, Firefox 57, Chrome 63, Safari 11 or later, Internet Explorer 11, MS Edge, Android 4.4 or later.



Note: Effective September 30, 2018, Adobe Scene7 Viewers is ending support of Transport Layer Security 1.0 (TLS 1.0). As such, Scene7 will no longer support viewers on the following browsers and/or platforms that support TLS 1.0. (Adobe Systems recommends the use of TLS 1.2 or later.)

- Android 2.3.7
- Android 4.0.4
- Android 4.1.1
- Android 4.2.2
- Android 4.3
- Internet Explorer 7 on Windows Vista
- Internet Explorer 8 on Windows XP
- Internet Explorer 8–10 on Windows 7
- Internet Explorer 10 on Windows Phone 8.0

- Safari 5.1.9 on Apple OS X 10.6.8
- Safari 6.0.4 on Apple OS X 10.8.4
- Java 6u45
- Java 7u25
- OpenSSL 0.9.8y
- Baidu January 2015



Note: *FLASH VIEWERS END-OF-LIFE—Effective January 31, 2017, Adobe Scene7 Publishing System officially ended support for the Flash viewer platform. For more information about this important change, see the following FAQ website:*

<https://docs.adobe.com/content/docs/en/aem/6-1/administer/integration/marketing-cloud/scene7/flash-eol.html>.

Installing the viewers on the same server

Instructions for installing the Scene7 Viewers API.

Install and test Image Serving before you install the Image Serving viewers.

Copy the IS Viewers files to your hard drive and then deploy the `s7viewers.war` file into the `../ImageServing/webapps` directory. See your Image Serving documentation for instructions on how to deploy, start, stop, and manage the Image Server.



Note: *There is no upgrade install for the Image Serving viewers. Adobe recommends that you back up any existing Scene7 viewers directory before you continue with the install.*

To install the viewers on the same server

1. Rename the viewer `.war` to the desired context and deploy file to the location you want.
2. Create a copy of `viewers-support.conf` and set the cache location.

Navigate to `.../imageserving/conf`. Make a copy of the `viewers-support.conf` file and name it as desired.

Open the newly created file and set `disk-cache.cache-root` to the cache location you want. The default is `cache/is-viewers`. Save and close the file.

3. Edit the context-specific configuration file to point to the newly created conf file.

Navigate to `.../imageserving/conf/Catalina/localhost`. Open for editing the XML file using the context set for the war file. In `<context>` node, add parameter `<Parameter name="com.scene7.is.viewers.configFile" value="<absolute path>/<context name.conf" />`. Save and close the file.

Example

```
<Context crossContext="true" >
  <Parameter name="com.scene7.is.viewers.configFile"
value="/usr/local/scene7/ImageServing/conf/viewers-support40.conf"/>
</Context>
```

4. Set `this.isViewerRoot` parameter in `config.js`.
Open `config.js` located in the root of the newly created viewer folder. Set parameter `this.isViewerRoot = "/s7viewers"` to the context of the war file (for example, `/s7viewers-4.0`). Save and close the file.

Compatibility notes

Compatibility notes for operating systems, browsers, and mobile devices.

Blackberry

- Incompatibility with older Adaptive Video Sets. You may need to re-upload Adaptive Video Sets to allow playback.

General

- Browser side scaling may cause UI and images to become blurry as user zooms into page. UI formatting may also display incorrectly depending upon zoom. This will carry over to full screen.
- Due to the size limitation on mobile devices the Mixed Media Viewer uses slide gesture to swap frames in embedded image sets instead of tapping the embedded swatches component. Component is there as a visual indicator.
- In Internet Explorer browsers and some touch devices, full-screen mode does not occupy the entire device screen. Instead, it resizes the application to the size of the browser window.
- Close button does not work under iOS 8.0 and iOS 8.1 but works under iOS 8.2.

Galaxy SIII

- Memory leak seen with Zoom and eCatalog viewers. Repeated navigation through frames may cause browser to crash.
- Double-tapping on a viewer may cause the entire page to zoom instead of just the viewer with browser-side scaling enabled.

Galaxy S4

- Device detected as tablet in portrait mode with Full Screen checked in browser settings.

Galaxy Nexus

- Double-tapping on a viewer may cause the entire page to zoom instead of just the viewer, with browser-side scaling enabled.

Galaxy Nexus 10 and Galaxy Tablet

- eCatalog shows incorrect page spread with portrait and landscape orientations.

HTC Mobile Devices

- Inability to disable native pinch-zoom is a "feature" of HTC UI wrapper (HTC Sense). This feature can cause an entire page to zoom when using "pinch to zoom" gesture on the viewer. Use a double-tap gesture instead.
- Image map icons may overlap if image maps are small and close together.

HTML5 Video Viewer

- Internet Explorer 9 – Custom Poster images do not display.
- IntialBitRate modifier is only supported with software HLS and flash HDS playback. It does not work when playback is using the native player.
- OGG and WebM progressive playback not supported.
- Browser scaling may cause the video player to display at an incorrect size (include Windows OS control panel Display settings).

- Video seek using HLS streaming on Safari may be inconsistent.

Internet Explorer

- Quirks mode is not supported.
- Compatibility mode is not supported.
- Internet Explorer on mobile is not supported.

iOS

- Large eCatalogs may cause the browser to crash on iPad 2.

Safari

- Safari 6.1 or later: Internet Plug-in settings may prevent Flash video playback.
- Video seek using HLS streaming on Safari may be inconsistent.
- Unable to seek to end of video on Safari 6 using HLS streaming.

Viewers release notes (5.10.1)

Adobe Scene7 Viewers

New features and enhancements for 5.10.1

- Added ARIA roles and attributes to support assistive technologies.
- Added "preload" modifier to video viewers to control video content preload.
- Removed support for flash streaming playback.

Bug fixes for 5.10.1

- Resolved an issue with IFRAME layout on iOS devices.
- Resolved XSS issues.

Known issues and restrictions for 5.10.1

- The Image Serving modifiers from IS commands are not added to the `req=set` request by design. However, modifiers that only affect image display work fine. Modifiers affecting size must be used in a complex asset. For example:

```
https://s7d9.scene7.com/s7viewers/html5/BasicZoomViewer.html?asset={Scene7SharedAssets/Backpack_B?extendn=0.5%252C0.5%252C0.5%252C0.5}
```

- Flyout viewer – Internet Explorer 9 sometimes remains onscreen after mouse off.
- Scaling the browser window leads to incorrect resizing.
- iPad 2 – Large eCatalog assets crash Safari on iOS.

All viewers

- Watermarks, obfuscation, and locking are not supported.
- Image presets are not supported.
- Adding or removing viewer from the DOM using `display:none` CSS or by dynamically detaching it from the parent node is not supported.
- Embedding a viewer in a table may result in incorrect sizing or placement of the viewer in non-native full-screen mode. Adobe recommends using DIVs instead.
- Parameters with explicit instance names in the code require instance names in the URL as well to be overwritten (for example, `zoomView.iconfeffect=0`).
- Image Serving command `crop` is not supported.
- Close button only works if the viewer is open in a child window.
- IS commands modifier does not support Image Serving modifiers that affect image size.

eCatalog

- Navigating to other HTML page and then returning occasionally causes the viewer to reset back to the first page.
- Page layout occasionally displays incorrectly after rotating the iOS device. Zooming into the page corrects the layout.
- Internal links only to left-most page in multi-page spreads. Affects mobile devices in portrait mode.
- Due to browser limitations, Print feature is not available in Internet Explorer 9.

Mixed Media

- Soundtrack play is not supported.

Social

- To render thumbnails properly in outgoing email, the `serverurl` modifier should have an absolute URL.

Video

- The poster image may encounter a `max_size` error. You may need to increase the limit setting for Image Serving Publish.
- Video captions require a company rule set if the hosting HTML page is served from an external server (not a Scene7 server). Contact technical support for assistance.
- Analytics tracking may report incorrect play percentage due to buffering.
- On iPad or Android devices, a black frame may show instead of a poster image.
- On iPad or Android devices, a black frame may flash onscreen during the loading of the viewer.
- On iPad devices, black borders are shown on the side of the VideoPlayer component when the background is set to white/transparent.
- On iPad, using iOS 7, the last frame of video may be distorted.
- On Chrome, Firefox, and Internet Explorer browsers, occasional macro blocking may occur during video seek in HLS streaming mode.
- The poster image may not show in the Microsoft Edge browser for the first time visitor.
- When progressive playback is used, the poster image may hide after the video loads in Internet Explorer 9.

Release notes archive

Scene7 Viewers 5.8.2 Release Notes

Adobe Scene7 Viewers

Bug fixes for 5.8.2

- User was unable to turn a page using mouse if `frametransition=turn`.
- `ImageMapEffect.mapTips=1` mouse click was generating a script error on touch-enabled devices.

New features and enhancements for 5.8.1

- Keyboard accessibility in Basic Zoom, Flyout, Mixed Media, Spin, Zoom, Video, Carousel, Interactive Image, and Interactive Video.
- Added video buffering icon to Video viewer, Mixed Media viewer, and Interactive Video viewer.
- The value `native` of `playback` modifier is deprecated; use the new value `progressive` instead.
- Added support for HTTPS streaming playback for Dynamic Media video.
- Added support for inline video playback on iPhone devices.
- Added support for external video playback to Video viewer.
- Improved swatches scroll behavior on touch devices.
- Added support for devices with both mouse and touch input running Internet Explorer 11 and Edge browser.
- Added ability to specify both left and right page with `initialframe` parameter in portrait mode on mobile devices in eCatalog viewer.

Bug fixes for 5.8.1

- User was unable to get focus on video controls by way of keyboard tabbing.
- In eCatalog viewer, single/double page layouts displayed incorrectly on some devices.

Known issues and restrictions for 5.8.2

- The Image Serving modifiers from IS commands are not added to the `req=set` request by design. However, modifiers that only affect image display work fine. Modifiers affecting size must be used in a complex asset. For example:

```
https://s7d9.scene7.com/s7viewers/html5/BasicZoomViewer.html?asset={Scene7SharedAssets/Backpack_B?extendn=0.5%252C0.5%252C0.5%252C0.5}
```

- Flyout viewer – Internet Explorer 9 sometimes remains onscreen after mouse off.
- Scaling the browser window leads to incorrect resizing.
- iPad 2 – A large eCatalog asset crashes Safari on iOS.

All viewers

- Watermarks, obfuscation, and locking are not supported.
- Image presets are not supported.
- Adding or removing viewer from the DOM using `display:none` CSS or by dynamically detaching it from the parent node is not supported.
- Embedding a viewer in a table may result in incorrect sizing or placement of the viewer in non-native full-screen mode. Adobe recommends using DIVs instead.
- Parameters with explicit instance names in the code require instance names in the URL as well to be overwritten (for example, `zoomView.iconfeffect=0`).
- Image Serving command `crop` is not supported.
- Close button only works if the viewer is open in a child window.
- IS commands modifier does not support Image Serving modifiers that affect image size.

eCatalog

- Navigating to other HTML page and then returning occasionally causes the viewer to reset back to the first page.
- Page layout occasionally displays incorrectly after rotating the iOS device. Zooming into the page corrects the layout.
- Internal links only to left-most page in multi-page spreads. Affects mobile devices in portrait mode.
- Due to browser limitations, Print feature is not available in Internet Explorer 9.

Mixed Media

- Soundtrack play is not supported.

Social

- To render thumbnails properly in outgoing email, the `serverurl` modifier should have an absolute URL.

Video

- The poster image may encounter a `max size` error. You may need to increase the limit setting for Image Serving Publish.
- Video captions require a company rule set if the hosting HTML page is served from an external server (not a Scene7 server). Contact technical support for assistance.
- Analytics tracking may report incorrect play percentage due to buffering.
- On iPad or Android devices, a black frame may show instead of a poster image.

- On iPad or Android devices, a black frame may flash onscreen during the loading of the viewer.
- On iPad devices, black borders are shown on the side of the VideoPlayer component when the background is set to white/transparent.
- On iPad, using iOS 7, the last frame of video may be distorted.
- On Chrome, Firefox, and Internet Explorer browsers, occasional macro blocking may occur during video seek in HLS streaming mode.
- The poster image may not show in the Microsoft Edge browser for the first time visitor.
- When progressive playback is used, the poster image may hide after the video loads in Internet Explorer 9.

Scene7 Viewers 5.5.2 Release Notes

Adobe Scene7 Viewers

Bug fixes for 5.5.2

- Video failed to play in Internet Explorer 11 on Windows 7.
- `initialframe` was not affecting portrait mode on mobile devices for eCatalog.

New features and enhancements for 5.5.1

- Added Adobe Marketing Cloud Org ID support to the Adobe Analytics integration.
- Updated AppMeasurement JavaScript library to version 1.6.1.
- Search capability now added to the eCatalog viewer.
- Added HLS streaming video playback as a default video delivery method for the majority of desktop systems. Flash-based HDS video streaming is still available as an alternative playback option.
- Added support for both mouse and touch input running under Chrome browser. Microsoft Surface devices fall under this category however, Microsoft Edge does not work with touch input with 5.5 viewers.

Bug fixes for 5.5.1

- Support for right-to-left orientation in eCatalog viewer.
- `tip=0,-1,0` was causing an out-of-range error.

Known issues and restrictions for 5.5.1

- The Image Serving modifiers from IS commands are not added to the `req=set` request by design. However, modifiers that only affect image display work fine. Modifiers affecting size must be used in a complex asset. For example:

```
https://s7d9.scene7.com/s7viewers/html5/BasicZoomViewer.html?asset={Scene7SharedAssets/Backpack_B?extendn=0.5%252C0.5%252C0.5%252C0.5}
```

- Flyout viewer – Internet Explorer 9 sometimes remains onscreen after mouse off.
- Scaling the browser window leads to incorrect resizing.
- iPad 2 – A large eCatalog asset crashes Safari on iOS.

All viewers

- Watermarks, obfuscation, and locking are not supported.
- Image presets are not supported.

- Adding or removing viewer from the DOM using `display:none` CSS or by dynamically detaching it from the parent node is not supported.
- Embedding a viewer in a table may result in incorrect sizing or placement of the viewer in non-native full-screen mode. Adobe recommends using DIVs instead.
- Parameters with explicit instance names in the code require instance names in the URL as well to be overwritten (for example, `zoomView.iconfeffect=0`).
- Image Serving command `crop` is not supported.
- Close button only works if the viewer is open in a child window.
- IS commands modifier does not support Image Serving modifiers that affect image size.

eCatalog viewer

- Navigating to other HTML page and then returning occasionally causes the viewer to reset back to the first page.
- Page layout occasionally displays incorrectly after rotating the iOS device. Zooming into the page corrects the layout.
- Internal links only to left-most page in multi-page spreads. Affects mobile devices in portrait mode.
- `InitialFrame` links only to left-most page in multi-page spreads. Affects mobile devices in portrait mode.
- Due to browser limitations, Print feature is not available in Internet Explorer 9.

Mixed Media viewer

- Soundtrack play is not supported.

Social viewer

- To render thumbnails properly in outgoing email, the `serverurl` modifier should have an absolute URL.

Video viewer

- The poster image may encounter a `max size` error. You may need to increase the limit setting for Image Serving Publish.
- Video captions require a company rule set if the hosting HTML page is served from an external server (not a Scene7 server). Contact technical support for assistance.
- Analytics tracking may report incorrect play percentage due to buffering.
- On iPad or Android devices, a black frame may show instead of a poster image.
- On iPad or Android devices, a black frame may flash onscreen during the loading of the viewer.
- On iPad devices, black borders are shown on the side of the VideoPlayer component when the background is set to white/transparent.
- On iPad, using iOS 7, the last frame of video may be distorted.
- On Chrome, Firefox, and Internet Explorer browsers, occasional macro blocking may occur during video seek in HLS streaming mode.
- The poster image may not show in the Microsoft Edge browser for the first time visitor.
- When progressive playback is used, the poster image may hide after the video loads in Internet Explorer 9.

Scene7 Viewers 5.4.2 Release Notes

Adobe Scene7 Viewers

Viewer upgrades are generally backwards compatible. With this release, a viewer change for namespace support was made. As such, all out-of-the-box viewer presets were updated to reflect this change. However, if you created your own custom viewer presets, your viewers may exhibit problems and require updating your CSS based on these known issues:

- Video viewer – Play/Pause button does not display Replay state.
- Video viewer – Navigation markers do not display.
- BasicZoom, Zoom, and Spin viewers – IconEffect does not display or displays SDK default art.
- Flyout and Zoom viewers – Swatch scroll buttons do not display for large sets.

It is recommended that you test viewers on our staging environment. The following website gives instructions on how you can set up your system to access our staging server:

<http://helpx.adobe.com/experience-manager/scene7/kb/base/upgrade-management/testing-scene7-viewers-upgrade.html>

After you have set up your computer to access the staging server, you can check your website to test the upgrade. For customers using out-of-the-box viewers, best practice is for you to test against our standard staging server and `s7is1-preview-staging.scene7.com`.

New features and enhancements for 5.4.2

- Viewer SDK is used in a namespace.
- Viewer `dispose()` API.
- Added support for Chrome browser on iOS devices.
- Added support for Microsoft Edge browser.
- Added "Favorites" feature to eCatalog viewer.
- Added support for running a viewer in the container which is hidden with `display:none` CSS or is detached from the DOM.
- Added ability to control horizontal and vertical spin sensitivity and lock the direction of spin.
- Viewer uses consolidated JavaScript file for SDK requests.

Bug fixes for 5.4.2

- Intermittent pinch-to-zoom non-functional.
- Full-screen buttons do not work on Microsoft Edge browser.
- Mode attribute on outer container is not reset to standard after leaving full-screen.
- Text-align property in parent DIV affecting button placement on toolbar.
- Flyout viewer: displaying the upper-left tile when the user scrolls the mouse over the main image.
- Combination of "initialFrame, direction" is causing the image map to display incorrectly.
- Unable to switch rows with an up/down gesture when a spinset contains only two rows.
- Native controls do not work.
- Chrome blocking flashproxy video playback.
- Zoom viewer and Mixed Media viewer: Swatches have a transparent background in simulated full-screen mode
- Viewers: second and subsequent call of `init()` should be ignored.

Known issues and restrictions for 5.4.2

- The Image Serving modifiers from iscommands are not added to the `req=set` request by design. Modifiers that only affect image display work fine. Modifiers affecting size must be used in a complex asset. For example:

```
https://s7d9.scene7.com/s7viewers/html5/BasicZoomViewer.html?asset={Scene7SharedAssets/Backpack_B?extendn=0.5%252C0.5%252C0.5%252C0.5}
```

- Flyout – Internet Explorer 9 sometimes remains onscreen after mouse off.
- ZoomViewer – Unable to zoom an image by mouse click (Windows 8, touch screen, Chrome).
- Browser scaling leads to wrong resizing.
- iPad 2 – Big eCatalog asset crashes Safari on iOS.

All Viewers

- Watermarks, obfuscation, and locking are not supported.
- Image presets are not supported.
- Adding or removing viewer from the DOM using `display:none` CSS or by dynamically detaching it from the parent node is not supported.

All viewers

- Embedding the viewer in a table may result in incorrect sizing or placement of the viewer in non-native full-screen mode. Use DIVs instead.
- Parameters with explicit instance names in the code require instance names in the URL to be overwritten as well. For example, `zoomView.iconeffect=0`.
- Image Serving command crop is not supported.
- Close button only works if the viewer is open in a child window.
- Iscommands modifier does not support Image Serving modifiers that affect image size.

eCatalog viewer

- Navigating to other HTML pages and then returning occasionally causes the viewer to reset back to the first page.
- The page layout occasionally displays incorrectly after rotating the iOS device. Zooming into the page corrects the layout.
- Internal links only to the leftmost page in multi-page spreads. This issue affects mobile devices in portrait mode.
- InitialFrame links only to the leftmost page in multi-page spreads. This issue affects mobile devices in portrait mode.
- Due to browser limitations, the Print feature is not available in Internet Explorer 9.

Mixed Media viewer

- Soundtrack play is not supported.

Social viewer

- To render thumbnails properly in outgoing email the `serverurl` modifier must use an absolute URL.

Video viewer

- The poster image may encounter a "max size" error. The user may need to increase the limit setting for Image Serving Publish.
- Video captions require a company ruleset if they are hosting an HTML page that is served from an external server that is not a Scene7 server. Contact Adobe technical support for assistance.

- Analytics tracking may report incorrect Play percentage due to buffering.
- Black frame instead of poster image may show on iPad or Android devices.
- Black frame may flash on screen during viewer load on iPad or Android devices.
- Black borders are shown on side of VideoPlayer component when background is set to white or transparent on iPad devices.
- The last frame of a video may be distorted on iPad using iOS 7.

Scene7 Viewers 5.2.3 Release Notes

Adobe Scene7 Viewers

Viewer upgrades are backwards compatible and therefore, no changes are necessary to your existing web code. However, it is recommended that you test the new viewers on our staging environment. The following website gives instructions on how you can set up your system to access our staging server:

<http://helpx.adobe.com/experience-manager/scene7/kb/base/upgrade-management/testing-scene7-viewers-upgrade.html>

After you have set up your computer to access the staging server, you can check your website to test the upgrade. For customers using out-of-the-box viewers, best practice is for you to test against our standard staging server and `s7is1-preview-staging.scene7.com`.

IMPORTANT: The latest version of Firefox now prevents the Adobe Flash plug-in from running on websites by default. If you have deployed the Scene7 video player that uses the Adobe Flash plug-in on desktop systems, you have the following three options:

- Users will see a prompt at the top of the web page they are viewing in Firefox. In the prompt, they can either choose to continue blocking the plug-in or allow the plug-in to run, in which case the video plays as usual.
- Set the video player to `playback=native` mode. Doing so plays the MP4 video progressively.
See [VideoPlayer.playback](#).
- Create an OGG version of the video which can be used as a fallback.

Adobe is investigating options to address this issue in Firefox in a future release.

New features and enhancements for 5.2.3

- Flyout Viewer incorrectly detects "legacy" sizing in case `s7flyoutzoomview` has padding.

New features and enhancements for 5.2.2

- Added support for Inline Zoom in the Mixed Media Viewer.
- Added support for Print, Download, and Favorites in the eCatalog Viewer.
- Added ability to retrieve ParameterManager using the `getComponent` API.
- Converted Spin Viewer, Zoom Viewer, Video Viewer, and Flyout Viewer to use sprites for artwork.
- Added support for Internet Explorer 11 native full-screen.
- Refactored simulated (non-native) full-screen support in Container.
- Increased CSS Small Marker size to support larger phones.
- Removed CSS size markers for Spin, BasicZoom, Zoom, Spin, and MixedMedia viewers on desktop browsers.
- Added support to allow quality configuration of preloaded frames in SpinView.

Bug fixes for 5.2.2

- Galaxy S4 – Portrait mode: Incorrect CSS size marker when full-screen mode enabled
- Internet Explorer 9 and Internet Explorer 10 – Zoom works incorrectly if the width property is defined for IMGs in the CSS.
- [eCatalog] Image maps are stuck after using Javascript templates for external linking on Internet Explorer 9.

Known issues and restrictions for 5.2.2

- The Image Serving modifiers from iscommands are not added to the `req=set` request by design. Modifiers that only affect image display work fine. Modifiers affecting size must be used in a complex asset. For example:

```
https://s7d9.scene7.com/s7viewers/html5/BasicZoomViewer.html?asset={Scene7SharedAssets/Backpack_B?extendn=0.5%252C0.5%252C0.5%252C0.5}
```

- iPad – Manual sets fail to display on Chrome browser
- Flyout – Internet Explorer 9 sometimes remains on-screen after mouse off.
- ZoomViewer – Unable to zoom an image by mouse click (Windows 8, touch screen, Chrome).
- Browser scaling leads to wrong resizing.
- iPad 2 – Big eCatalog asset will crash Safari on IOS

All Scene7 viewers

- Watermarks, obfuscation, and locking are not supported.
- Image presets are not supported.
- Adding or removing viewer from the DOM using `display:none` CSS or by dynamically detaching it from the parent node is not supported.

All viewers

- Embedding the viewer in a table may result in incorrect sizing or placement of the viewer in non-native full-screen mode. Use DIVs instead.
- Parameters with explicit instance names in the code require instance names in the URL to be overwritten as well. For example, `zoomView.iconeffect=0`.
- Image Serving command crop is not supported.
- Close button only works if the viewer is open in a child window.
- Iscommands modifier does not support Image Serving modifiers that affect image size.
- CSS style `display:none` on DIV container is not supported. This includes JQuery `hide()` method.

eCatalog viewer

- Navigating to other HTML pages and then returning occasionally causes the viewer to reset back to the first page.
- The page layout occasionally displays incorrectly after rotating the iOS device. Zooming into the page corrects the layout.
- Internal links only to the left-most page in multi-page spreads. This issue affects mobile devices in portrait mode.
- InitialFrame links only to the left-most page in multi-page spreads. This issue affects mobile devices in portrait mode.
- Due to browser limitations, the Print feature is not available in Internet Explorer 9.

Mixed Media viewer

- Sound track play is not supported.

Social viewer

- To render thumbnails properly in outgoing email the `serverurl` modifier must use an absolute URL.

Video viewer

- The poster image may encounter "max size" error. The user may need to increase the limit setting for Image Serving Publish.
- Video captions require a company ruleset if they are hosting an HTML page that is served from an external server that is not a Scene7 server. Contact Adobe technical support for assistance.
- Analytics tracking may report incorrect play percentage due to buffering
- Black frame instead of poster image may show on iPad or Android devices.
- Black frame may flash on screen during viewer load on iPad or Android devices.
- Black borders are shown on side of VideoPlayer component when background is set to white or transparent on iPad devices.
- Last frame of video may be distorted on iPad using iOS 7.

Scene7 Viewers 5.2.2 Release Notes

Adobe Scene7 Viewers

Viewer upgrades are backwards compatible and therefore, no changes are necessary to your existing web code. However, it is recommended that you test the new viewers on our staging environment. The following website gives instructions on how you can set up your system to access our staging server:

<http://helpx.adobe.com/experience-manager/scene7/kb/base/upgrade-management/testing-scene7-viewers-upgrade.html>

After you have set up your computer to access the staging server, you can check your website to test the upgrade. For customers using out-of-the-box viewers, best practice is for you to test against our standard staging server and `s7is1-preview-staging.scene7.com`.

New features and enhancements

- Added support for Inline Zoom in the Mixed Media Viewer.
- Added support for Print, Download, and Favorites in the eCatalog Viewer.
- Added ability to retrieve ParameterManager using the `getComponent` API.
- Converted Spin Viewer, Zoom Viewer, Video Viewer, and Flyout Viewer to use sprites for artwork.
- Added support for Internet Explorer 11 native full-screen.
- Refactored simulated (non-native) full-screen support in Container.
- Increased CSS Small Marker size to support larger phones.
- Removed CSS size markers for Spin, BasicZoom, Zoom, Spin, and MixedMedia viewers on desktop browsers.
- Added support to allow quality configuration of preloaded frames in SpinView.

Bug fixes

- Galaxy S4 – Portrait mode: Incorrect CSS size marker when full-screen mode enabled
- Internet Explorer 9 and Internet Explorer 10 – Zoom works incorrectly if the width property is defined for IMGs in the CSS.
- [eCatalog] Image maps are stuck after using Javascript templates for external linking on Internet Explorer 9.

Known issues and restrictions

- The Image Serving modifiers from iscommands are not added to the `req=set` request by design. Modifiers that only affect image display work fine. Modifiers affecting size must be used in a complex asset. For example:

```
https://s7d9.scene7.com/s7viewers/html5/BasicZoomViewer.html?asset={Scene7SharedAssets/Backpack_B?extendn=0.5%252C0.5%252C0.5%252C0.5}
```

- iPad – Manual sets fail to display on Chrome browser
- Flyout – Internet Explorer 9 sometimes remains on-screen after mouse off.
- ZoomViewer – Unable to zoom an image by mouse click (Windows 8, touch screen, Chrome).
- Browser scaling leads to wrong resizing.
- iPad 2 – Big eCatalog asset will crash Safari on IOS

All Scene7 viewers

- Watermarks, obfuscation, and locking are not supported.
- Image presets are not supported.
- Adding or removing viewer from the DOM using `display:none` CSS or by dynamically detaching it from the parent node is not supported.

All viewers

- Embedding the viewer in a table may result in incorrect sizing or placement of the viewer in non-native full-screen mode. Use DIVs instead.
- Parameters with explicit instance names in the code require instance names in the URL to be overwritten as well. For example, `zoomView.iconeffect=0`.
- Image Serving command crop is not supported.
- Close button only works if the viewer is open in a child window.
- Iscommands modifier does not support Image Serving modifiers that affect image size.
- CSS style `display:none` on DIV container is not supported. This includes JQuery `hide()` method.

eCatalog viewer

- Navigating to other HTML pages and then returning occasionally causes the viewer to reset back to the first page.
- The page layout occasionally displays incorrectly after rotating the iOS device. Zooming into the page corrects the layout.
- Internal links only to the left-most page in multi-page spreads. This issue affects mobile devices in portrait mode.
- InitialFrame links only to the left-most page in multi-page spreads. This issue affects mobile devices in portrait mode.
- Due to browser limitations, the Print feature is not available in Internet Explorer 9.

Mixed Media viewer

- Sound track play is not supported.

Social viewer

- To render thumbnails properly in outgoing email the `serverurl` modifier must use an absolute URL.

Video viewer

- The poster image may encounter "max size" error. The user may need to increase the limit setting for Image Serving Publish.

- Video captions require a company ruleset if they are hosting an HTML page that is served from an external server that is not a Scene7 server. Contact Adobe technical support for assistance.
- Analytics tracking may report incorrect play percentage due to buffering
- Black frame instead of poster image may show on iPad or Android devices.
- Black frame may flash on screen during viewer load on iPad or Android devices.
- Black borders are shown on side of VideoPlayer component when background is set to white or transparent on iPad devices.
- Last frame of video may be distorted on iPad using iOS 7.

Scene7 Viewers 5.1.1 Release Notes

Adobe Scene7 Viewers

Viewer upgrades are backwards compatible and therefore, no changes are necessary to your existing web code. However, it is recommended that you test the new viewers on our staging environment. The following website gives instructions on how you can set up your system to access our staging server:

<http://helpx.adobe.com/experience-manager/scene7/kb/base/upgrade-management/testing-scene7-viewers-upgrade.html>

After you have set up your computer to access the staging server, you can check your website to test the upgrade. For customers using out-of-the-box viewers, best practice is for you to test against our standard staging server and `s7is1-preview-staging.scene7.com`.

New features and enhancements

- Updated support for Internet Explorer 11
- Changed viewers to use Track requests instead of Page requests for Adobe Analytics tracking
- Added support for responsive images using image commands and/or image presets based upon image width
- Added support for passing localization text strings to JSON argument of viewers
- Added support for native page scrolling in viewer swatches
- Improved support for complex assets and image templates to viewers
- Added support for table of contents to eCatalog viewer
- Added support for image map overlays versus icons to eCatalog
- Added support for page turn animation to eCatalog viewer
- Added support for native page scrolling in eCatalog
- Added support for displaying narrow images using highlightmode modifier to Flyout Viewer
- Added support to control upscaling of images using zoomfactor modifier to Flyout Viewer
- Added support for tracking events generated by Flyout Viewer
- Added caption support for video (first video only) to MixedMedia viewer
- Increased initial bitrate default to 1400 for Video and MixedMedia viewers
- Added support for looping video playback to Video viewer
- Added support for navigation chapters to Video viewer

Bug fixes

- eCatalog Viewer not displaying PageIndicator on iPhone
- Images fail to display if Image Serving modifier with comma is appended to asset ID
- Imagesets: Image Serving modifiers appended to assetID are ignored
- Internet Explorer 9: s7sdk.event: PageMouseEvent: click and double-click events triggers crash browser page
- Internet Explorer 9 and Internet Explorer 10: eCatalog viewer breaks after activating image map with JavaScript template
- MixedMedia viewer: Zoom button states not reset when swapping asset types
- MixedMedia viewer: Viewer collapsed in responsive mode if first asset is 2dspinset
- Firefox: Broken image icons briefly displayed on flyout until tile(s) are loaded
- Pinch gestures do not generate zoom
- eCatalog: Imagemaps: JavaScript error occurring with relative links

Known issues and restrictions

All Scene7 viewers

- Watermarks, obfuscation, and locking are not supported.
- Image presets are not supported.

All viewers

- Embedding viewer in table may result in incorrect sizing or placement of viewer in non-native fullscreen mode. Suggest using DIVs instead.
- Parameters with explicit instance names in the code must be overwritten; instance names in a URL must also be overwritten. For example, `zoomView.iconeffect=0`.
- Image Serving command crop is not supported.
- Close button only works if the viewer is open in a child window.
- Iscommands modifier does not support Image Serving modifiers that affect image size.
- CSS style "display: none" on DIV container is not supported at this time. This includes JQuery hide() method.

eCatalog viewer

- Navigating to another HTML page and then returning occasionally causes the viewer to reset back to the first page.
- Page layout occasionally displays incorrectly after rotating the iOS device. Zoom into page corrects layout.
- Internal links only to leftmost page in multi-page spreads. Affects mobile devices in portrait mode.
- InitialFrame links only to leftmost page in multi-page spreads. Affects mobile devices in portrait mode.

Mixed Media viewer

- Sound track play is not supported.

Social viewer

- To render thumbnails properly in outgoing email the `serverurl` modifier should have an absolute URL.

Video viewer

- The poster image may encounter "max size" error. The user may need to increase the limit setting for Image Serving Publish.

- Video captions require a company ruleset if they are hosting an HTML page that is served from an external server that is not a Scene7 server. Contact Adobe technical support for assistance.
- Analytics tracking may report incorrect play percentage due to buffering
- Black frame instead of poster image may show on iPad or Android devices.
- Black frame may flash on screen during viewer load on iPad or Android devices.
- Black borders are shown on side of VideoPlayer component when background is set to white/transparent on iPad devices.
- Last frame of video may be distorted on iPad using iOS 7.

Scene7 Viewers 5.0.1 Release Notes

Adobe Scene7 Viewers

Viewer upgrades are backwards compatible and therefore, no changes are necessary to your existing web code. However, it is recommended that you test the new viewers on our staging environment. The following website gives instructions on how you can set up your system to access our staging server:

<http://helpx.adobe.com/experience-manager/scene7/kb/base/upgrade-management/testing-scene7-viewers-upgrade.html>

After you have set up your computer to access the staging server, you can check your website to test the upgrade. For customers using out-of-the-box viewers, best practice is for you to test against our standard staging server and `s7is1-preview-staging.scene7.com`.

New features and changes

The following were updated:

- Minimum system requirement for Windows operating system is now Windows 7.
- Minimum system requirement for Internet Explorer is now version 9.
- Minimum system requirement for Mac OS is now Mac OS 10.8.
- Minimum system requirement for Safari is now version 6.

The following were refactored:

- viewers to support responsive design.
- viewers to use CSS input and size markers to control display on desktop and mobile devices.

The following is now supported:

- Adobe Analytics video reports to Video Viewer and Mixed Media Viewer.
- Event tracking of multiple viewers on the same page.
- Added `SetIndicator` to Video Viewer and Mixed Media Viewer to visually represent swatches on mobile devices.
- Added `SetIndicator` to MixedMediaViewer to visually represent swatches on mobile.
- Added `getComponent` API to allow external access to viewer components.
- viewers for configuration parameters to be passed as a single JSON object.
- WebVTT captions to be passed as JSON object, eliminating the need for ruleset configuration.
- JavaScript templates in eCatalog Viewer for linking image maps to external pages.
- Linking to pages within the eCatalog Viewer.
- `stagesize` to Flyout Viewer.
- `initComplete` handler to viewers for triggering functions after viewers.

- Analytics handler to viewers for supporting third-party tracking.
- Added "Select All" button to make it easier to obtain Link and Embed Code for social sharing.
- FlyoutZoomView for resize method.
- FlyoutZoomView for setting image reload breakpoints when used for responsive design.
- Added click support for adjusting volume.
- SpinView and ZoomView to apply gestures to embedding web page when not interacting with viewer.
- Added `unload()` API to SpinView and ZoomView to allow disposing of the component at runtime.
- The Video Viewer to apply gestures to the embedding web page when not interacting with viewer.
- The ZoomView for initial zoom region.
- ZoomView to hide `IconEffect` component when zoom is not possible

The following bugs were fixed:

- Potential Flash Fastzoom and Flyout viewers security issues.
- viewers not loading due to unrecognized UserAgent.
- eCatalog Viewer mouse drag-and-drop not working in Internet Explorer with page containing image maps.
- Video Viewer prevents access to unblock Flash plug-in.
- viewers prevent printing on Firefox browser.
- Video Viewer displays a slight jitter when initiating or resetting zoom.
- HTTPS support for HTML5 and AS3 viewers.

The following were removed:

- Social features displaying in full screen mode on viewers.
- Close button displaying in full screen mode on viewers.

Known issues and restrictions

All Scene7 viewers

- Watermarks, obfuscation, and locking are not supported.

All viewers

- Embedding viewer in table may result in incorrect sizing or placement of viewer in non-native fullscreen mode. Suggest using DIVs instead.
- Parameters with explicit instance names in the code must be overwritten; instance names in a URL must also be overwritten. For example, `zoomView.iconeffect=0`.
- Image Serving command crop is not supported.
- Close button only works if the viewer is open in a child window.

Adobe Analytics Training

- When using Scene7 viewers with Adobe Analytics tracking based on `s_code.jsp`, "page" requests are sent rather than "track" requests for all tracked events. This results in over-inflated page view metrics for pages that contain a Scene7 viewer. To fix this issue, copy the contents of `AppMeasurementBridge.jsp` to a Javascript file to be served with the viewer page. In the resultant Javascript replace all calls to `s.t()` with `s.tl()` for all events other than LOAD event. This issue does not impact Scene7 video viewers using the new Adobe Analytics Video Heartbeat reporting.
- Internet Explorer: "Access is denied" JavaScript error occurs using SSL / HTTPS with Adobe Analytics tracking enabled.

eCatalog viewer

- JavaScript templates in image maps are not supported.
- Navigating to another HTML page and then returning occasionally causes the viewer to reset back to the first page.
- Set `ImageMapEffect` rollover modifier to 1 to invoke infopanel.
- `Frametransition` set to `none` or `fade` is not supported.
- Page layout occasionally displays incorrectly after rotating the iOS device. Zoom into page corrects layout.
- Internal links only to leftmost page in multi-page spreads. Affects mobile devices in portrait mode.
- `InitialFrame` links only to leftmost page in multi-page spreads. Affects mobile devices in portrait mode.

Mixed Media viewer

- Sound track play is not supported at this time.

Social viewer

- To render thumbnails properly in outgoing email the `serverurl` modifier should have an absolute URL.

Video viewer

- The poster image may encounter "max size" error. The user may need to increase the limit setting for Image Serving Publish.
- Video captions require a company ruleset if they are hosting an HTML page that is served from an external server that is not a Scene7 server. Contact Adobe technical support for assistance.
- Analytics tracking may report incorrect play percentage due to buffering
- Black frame instead of poster image may show on iPad or Android devices.
- Black frame may flash on screen during viewer load on iPad or Android devices.
- Black borders are shown on side of VideoPlayer component when background is set to white/transparent on iPad devices.
- Last frame of video may be distorted on iPad using iOS 7.

Scene7 Viewers 4.9.2 Release Notes

Enhancements, bug fixes, and known issues in Scene7 Viewers 4.9.2

Enhancements in Scene7 Viewers 4.9.2

Viewer upgrades are backwards compatible and therefore, no changes are necessary to your existing web code. However, it is recommended that you test the new viewers on our staging environment. Contact Technical Support for instructions on how to access the staging server so you can test your viewers. After this is done, you can check your website to test the upgrades.

- Increased minimum requirements for viewers to iOS6.
- Added support for custom event tracking to viewers.
- Added support to set the initial bit rate for Video viewer.
- Video viewer now defaults to use HLS streaming on Safari desktop.
- Refactored tooltips to address various bugs.
- Removed social feature restrictions on Video and eCatalog viewers from mobile devices.

Bug fixes in Scene7 Viewers 4.9.2

The following bugs are now fixed:

- viewers were displaying at the incorrect size after returning from full-screen and swapping assets.
- viewers were displaying close button in Full-screen mode.
- eCatalog viewer were not displaying image maps in portrait mode on mobile devices.

- eCatalog viewer was displaying pan buttons on mobile phones.
- eCatalog viewer default tool tips did not apply if the container id is not "ecatalog".
- eCatalog viewer Tooltips were hidden behind thumbnails in grid view.
- eCatalog viewer was displaying a page divider for single page.
- eCatalog viewer image maps failed to function on IE9.
- Mixed Media viewer was resetting the video scrubber position after resizing the viewer.
- Mixed Media viewer was using the incorrect art for spin pan buttons.
- Mixed Media viewer video was showing tooltips under mixed media swatches.
- HTML5 Mixed Media viewer spin buttons were displaying on tablet devices.
- Spin and Zoom viewer tooltips were clipped by edge in embedded viewer.
- Social Share tooltip position was displaced for the social buttons.
- Social Share tooltips format were not matching viewer tooltips.
- Tooltips did not display in full-screen mode on Mac OS with Safari 5.
- Video viewer incorrect size occurred for progressbar when rotating iPad and switching between screen modes.
- Video viewer was generating console logs by default.

Known issues in Scene7 Viewers 4.9.2

All Scene7 viewers

- Watermarks, obfuscation, and locking are not supported.

All viewers

- Parameters with explicit instance names in the code must be overwritten; instance names in a URL must also be overwritten.
For example, `zoomView.iconeffect=0`.
- Image Serving command crop is not supported.
- Close button only works if the viewer is open in a child window.
- To customize the tooltip format you add ! IMPORTANT to the CSS declaration.

eCatalog viewer

- JavaScript templates in image maps are not supported.
- Navigating to another HTML page and then returning occasionally causes the viewer to reset back to the first page.
- Set `ImageMapEffect` rollover modifier to 1 to invoke infopanel.
- `Frametransition` set to none or fade is not supported.
- Page layout occasionally displays incorrectly after rotating the iOS device. Zoom into page corrects layout.

Mixed Media viewer

- Sound track play is not supported at this time.

Social viewer

- To render thumbnails properly in outgoing email the `serverurl` modifier should have an absolute URL.

Video viewer

- The poster image may encounter "max size" error. The user may need to increase the limit setting for Image Serving Publish.
- Video captions require a company ruleset if they are hosting an HTML page that is served from an external server that is not a Scene7 server. Contact Adobe technical support for assistance.

Flash AS3—all viewers

- Double encode the # character in asset names.
- Server Support fails to load SWF animations with embedded videos.
- Server Support fails to load viewer skins if compiled for Flash Player 6. Workaround is to compile for Flash Player 7.
- Macintosh OS and Flash Player version 10,0,32,18: Workaround depends on a JavaScript bridge instead of a LocalConnection to communicate between Flex and Flash. So, the Flex application must be imbedded in the HTML wrapper.
- Currently the Flash Viewers support SWFs compiled for Flash 7 only.
- Because of an issue with Java v.1.5.0_06 the server component is not supported with that version of Java.
http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6274990.
- For a skin custom URL parameter, the parameter in the Skin URL must be URL encoded.
- Image Sets with extended ASCII characters in the name must be URL encoded twice when sent to the viewer or the FVCTX request fails because the Flash player decodes the name before passing it to the viewer code.
- If the embedding HTML page has significant content after the viewer, it is possible that the viewer may call a JavaScript function before the page is fully loaded. This may abort page loading. A suggested workaround is to use an onLoad event handler to delay initializing the viewer until after the page load is complete.
- The changes to the `S7Config.setFlashParam()` allow for more than one parameter/value to be passed with this syntax `S7Config.setFlashParam(<viewer ID>, <parameter name>, <parameter value> [, <parameter name>, <parameter value>])`, but in that case the first parameter to be passed must be the "image" parameter followed by the "currentFrame" parameter (if required), and then any other parameters.
- When using the `skipFrames=frameList` parameter to omit images, the viewer returns an error if an omitted frame is called directly either by the JavaScript change image function, `InitialFrame`, or in the case of a `RenderSet` that has two or more swatches associated with an image - when the swatch combination referencing the omitted frame is selected.
- Crop is not supported in conjunction with zoom targets or swatches in either the `image=` or `modifier=` arguments.
- The `rgn` argument is not supported for the flash viewers for image modifier.
- A new IE security patch requires the user to activate Flash and other ActiveX applications in IE. Please see the Microsoft site for information about workarounds.
- Security issues prevent the viewer from reading from a different server on the same domain unless both URLs are fully qualified domain names or the "cross domain policy" is specifically set to allow access from that server.
- With Flash 7's new security features, if the viewers attempt to read from a different server on the same domain from where the viewers are located (that is, they are redirected by the paths in the `infoServerUrl`, `searchEngineUrl`, or the `serverUrl` parameters) a security alert notifies the user of this fact and asks if you want to allow this, unless the "cross domain policy" is set to allow access from that server.

For more information on how to set up a "cross domain policy" see the following article:

http://kb2.adobe.com/cps/142/tn_14213.html.

- Exception error generated when viewer is embedded in HTTPS page. Issue is due to communication with server logging. For more information: <http://helpx.adobe.com/flash-player/kb/flash-player-issues-secure-sockets.html>

Viewers for AEM Assets and Scene7

The following viewers work under Adobe Experience Manager Assets and also under Scene7 Publishing System.

Basic Zoom

Basic Zoom Viewer is an image viewer that displays a single zoomable image. It has zoom tools, full screen support, and an optional close button. This viewer is the most lightweight. It is designed to work on desktops and mobile devices.



Note: Images that use IR (Image Rendering) or UGC (User Generated Content) are not supported by this viewer.

Viewer type 501.

See [System requirements](#).

Demo URL

https://s7d9.scene7.com/s7viewers/html5/BasicZoomViewer.html?asset=Scene7SharedAssets/Backpack_B

Using Basic Zoom Viewer

Basic Zoom Viewer represents a main JavaScript file and a set of helper files (a single JavaScript include with all the Viewer SDK components used by this particular viewer, assets, CSS) that the viewers downloads at runtime.

You can use Basic Zoom Viewer in pop-up mode using a production-ready HTML page provided with IS-Viewers or in embedded mode, where it is integrated into target web page using documented API.

Configuration and skinning are similar to that of the other viewers. All skinning is achieved by way of custom CSS.

See [Command reference common to all viewers – Configuration attributes](#) and [Command reference common to all Viewers – URL](#)

Interacting with Basic Zoom Viewer

Basic Zoom Viewer supports the following touch gestures that are common in other mobile applications.

When the viewer cannot process a user's swipe gesture it forwards the event to the web browser to perform a native page scroll. This kind of functionality lets the user navigate through the page even if the viewer occupies most of the device's screen area.

Gesture	Description
Single tap	Hides or reveals user interface elements.
Double tap	Zooms in one level until maximum magnification is reached. The next double tap gesture resets the viewer to the initial viewing state.
Pinch	Zooms in or out.
Swipe	If the image is in a reset state, the gesture performs a native page scroll.

Gesture	Description
	When the image is zoomed in, it moves the image. If the image is moved to the view edge and a swipe is performed in that direction, the gesture performs a native page scroll.

The viewer also supports both touch input and mouse input on Windows devices with touch screen and mouse. This support, however, is limited to Chrome, Internet Explorer 11, and Edge web browsers only.

This viewer is fully keyboard accessible.

See [Keyboard accessibility and navigation](#).

Embedding Basic Zoom Viewer

Different web pages have different needs for viewer behavior. Sometimes a web page provides a link that, when clicked, opens the viewer in a separate browser window. In other cases, it is necessary to embed the viewer right in the hosting page. In the latter case, the web page may have a static page layout, or use responsive design that displays differently on different devices or for different browser window sizes. To accommodate these needs, the viewer supports three primary operation modes: pop-up, fixed size embedding, and responsive design embedding.

About pop-up mode

In pop-up mode, the viewer is opened in a separate web browser window or tab. It takes the entire browser window area and adjusts in case the browser is resized or the device orientation is changed.

Pop-up mode is the most common for mobile devices. The web page loads the viewer using the `window.open()` JavaScript call, properly configured A HTML element, or any other suitable method.

It is recommended that you use an out-of-the-box HTML page for pop-up operation mode. In this case, it is called `BasicZoomViewer.html` and is located within the `html5/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/BasicZoomViewer.html
```

You can achieve visual customization by applying custom CSS.

The following is an example of HTML code that opens the viewer in a new window:

```
<a
href="http://s7d1.scene7.com/s7viewers/html5/BasicZoomViewer.html?asset=Scene7SharedAssets/Backpack_B"
target="_blank">Open popup viewer</a>
```

About fixed size embedding mode and responsive design embedding mode

In the embedded mode, the viewer is added to the existing web page, which may already have some customer content not related to the viewer. The viewer normally occupies only a part of a web page's real estate.

The primary use cases are web pages oriented for desktops or tablet devices, and also responsive designed pages that adjust layout automatically depending on the device type.

Fixed size embedding is used when the viewer does not change its size after initial load. This is the best choice for web pages that have a static layout.

Responsive design embedding assumes that the viewer may need to resize at runtime in response to the size change of its container DIV. The most common use case is adding a viewer to a web page that uses a flexible page layout.

In responsive design embedding mode, the viewer behaves differently depending on the way web page sizes its container `DIV`. If the web page sets only the width of the container `DIV`, leaving its height unrestricted, the viewer automatically chooses its height according to the aspect ratio of the asset that is used. This functionality ensures that the asset fits perfectly into the view without any padding on the sides. This use case is the most common for web pages using responsive web design layout frameworks like Bootstrap, Foundation, and so on.

Otherwise, if the web page sets both the width and the height for the viewer's container `DIV`, the viewer fills just that area and follows the size that the web page layout provides. A good example is embedding the viewer into a modal overlay, where the overlay is sized according to web browser window size.

Fixed size embedding

You add the viewer to a web page by doing the following:

1. Adding the viewer JavaScript file to your web page.
2. Defining the container `DIV`.
3. Setting the viewer size.
4. Creating and initializing the viewer.

1. Adding the viewer JavaScript file to your web page.

Creating a viewer requires that you add a script tag in the HTML head. Before you can use the viewer API, be sure that you include `BasicZoomViewer.js`. The `BasicZoomViewer.js` file is located under the `html5/js/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/js/BasicZoomViewer.js
```

You can use a relative path if the viewer is deployed on one of the Adobe Scene7 servers and it is served from the same domain. Otherwise, you specify a full path to one of Adobe Scene7 servers that have the IS-Viewers installed.

The relative path looks like the following:

```
<script language="javascript" type="text/javascript"
src="/s7viewers/html5/js/BasicZoomViewer.js"></script>
```



Note: You should only reference the main viewer JavaScript `include` file on your page. You should not reference any additional JavaScript files in the web page code which might be downloaded by the viewer's logic in runtime. In particular, do not directly reference `HTML5 SDK Utils.js` library loaded by the viewer from `/s7viewers` context path (so-called consolidated SDK `include`). The reason is that the location of `Utils.js` or similar runtime viewer libraries is fully managed by the viewer's logic and the location changes between viewer releases. Adobe does not keep older versions of secondary viewer `includes` on the server.

As a result, putting a direct reference to any secondary JavaScript `include` used by the viewer on the page breaks the viewer functionality in the future when a new product version is deployed.

2. Defining the container `DIV`.

Add an empty `DIV` element to the page where you want the viewer to appear. The `DIV` element must have its ID defined because this ID is passed later to the viewer API. The `DIV` has its size specified through CSS.

The placeholder `DIV` is a positioned element, meaning that the `position` CSS property is set to `relative` or `absolute`.

The following is an example of a defined placeholder `DIV` element:

```
<div id="s7viewer" style="position:relative"></div>
```


3. Setting the viewer size

You can set the static size for the viewer by either declaring it for `.s7basiczoomviewer` top-level CSS class in absolute units, or by using `stagesize` modifier.

You can put sizing in CSS directly on the HTML page, or in a custom viewer CSS file, which is then later assigned to a viewer preset record in Scene7 Publishing System, or passed explicitly using a style command.

See [Customizing Basic Zoom Viewer](#) for more information about styling the viewer with CSS.

The following is an example of defining a static viewer size in HTML page:

```
#s7viewer.s7basiczoomviewer {
  width: 640px;
  height: 480px;
}
```

You can set the `stagesize` modifier either in the viewer preset record in Scene7 Publishing System, or pass it explicitly with the viewer initialization code with `params` collection, or as an API call as described in the Command Reference section, like the following:

```
basicZoomViewer.setParam("stagesize", "640,480");
```

A CSS-based approach is recommended and is used in this example.

4. Creating and initializing the viewer.

When you have completed the steps above, you create an instance of `s7viewers.BasicZoomViewer` class, pass all configuration information to its constructor, and call `init()` method on a viewer instance. Configuration information is passed to the constructor as a JSON object. At minimum, this object should have `containerId` field which holds the name of viewer container ID and nested `params` JSON object with configuration parameters supported by the viewer. In this case, the `params` object must have at least the Image Serving URL passed as `serverUrl` property, and the initial asset as `asset` parameter. The JSON-based initialization API lets you create and start the viewer with a single line of code.

It is important to have the viewer container added to the DOM so that the viewer code can find the container element by its ID. Some browsers delay building DOM until the end of the web page. For maximum compatibility, call the `init()` method just before the closing BODY tag, or on the `body onload()` event.

At the same time, the container element should not necessarily be part of the web page layout just yet. For example, it may be hidden using `display:none` style assigned to it. In this case, the viewer delays its initialization process until the moment when the web page brings the container element back to the layout. When this occurs, the viewer load resumes automatically.

The following is an example of creating a viewer instance, passing minimum necessary configuration options to the constructor and calling the `init()` method. The example assumes `basicZoomViewer` is the viewer instance; `s7viewer` is the name of placeholder DIV; `http://s7d1.scene7.com/is/image/` is the Image Serving URL, and `Scene7SharedAssets/Backpack_B` is the asset:

```
<script type="text/javascript">
var basicZoomViewer = new s7viewers.BasicZoomViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/Backpack_B",
    "serverurl": "http://s7d1.scene7.com/is/image/"
  }
}).init();
</script>
```

The following code is a complete example of a trivial web page that embeds the Basic Zoom Viewer with a fixed size:

```
<!DOCTYPE html>
<html>
```

```

<head>
<script type="text/javascript"
src="http://s7d1.scene7.com/s7viewers/html5/js/BasicZoomViewer.js"></script>
<style type="text/css">
#s7viewer.s7basiczoomviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var basicZoomViewer = new s7viewers.BasicZoomViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/Backpack_B",
    "serverurl": "http://s7d1.scene7.com/is/image/"
  }
}).init();
</script>
</body>
</html>

```

Responsive design embedding with unrestricted height

With responsive design embedding, the web page normally has some kind of flexible layout in place that dictates the runtime size of the viewer's container DIV. For the following example, assume that the web page allows the viewer's container DIV to take 40% of the web browser window size, leaving its height unrestricted. The web page HTML code would look like the following:

```

<!DOCTYPE html>
<html>
<head>
<style type="text/css">
.holder {
  width: 40%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>

```

Adding the viewer to such a page is similar to the steps for fixed size embedding. The only difference is that you do not need to explicitly define the viewer size.

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Creating and initializing the viewer.

All the steps above are the same as with the fixed size embedding. Add the container DIV to the existing "holder" DIV. The following code is a complete example. Notice how viewer size changes when the browser is resized, and how the viewer aspect ratio matches the asset.

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7d1.scene7.com/s7viewers/html5/js/BasicZoomViewer.js"></script>
<style type="text/css">
.holder {
  width: 40%;
}

```

```

</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var basicZoomViewer = new s7viewers.BasicZoomViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Scene7SharedAssets/Backpack_B",
    "serverurl":"http://s7dl.scene7.com/is/image/"
  }
}).init();
</script>
</body>
</html>

```

The following examples page illustrates more real-life uses of responsive design embedding with unrestricted height:

https://marketing.adobe.com/resources/help/en_US/s7/vlist/vlist.html

Flexible size Embedding with Width and Height Defined

In case of flexible-size embedding with width and height defined, the web page styling is different. It provides both sizes to the "holder" DIV and center it in the browser window. Also, the web page sets the size of the HTML and BODY element to 100 percent.

```

<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>

```

The rest of the embedding steps are identical to the steps used for responsive embedding with unrestricted height. The resulting example is the following:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/BasicZoomViewer.js"></script>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;

```

```
height: 60%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var basicZoomViewer = new s7viewers.BasicZoomViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Scene7SharedAssets/Backpack_B",
    "serverurl":"http://s7dl.scene7.com/is/image/"
  }
}).init();
</script>
</body>
</html>
```

Embedding Using Setter-based API

Instead of using JSON-based initialization, it is possible to use setter-based API and no-args constructor. Using this API constructor does not take any parameters and configuration parameters are specified using `setContainerId()`, `setParam()`, and `setAsset()` API methods with separate JavaScript calls.

The following example illustrates using fixed size embedding with setter-based API:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/BasicZoomViewer.js"></script>
<style type="text/css">
#s7viewer.s7basiczoomviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var basicZoomViewer = new s7viewers.BasicZoomViewer();
basicZoomViewer.setContainerId("s7viewer");
basicZoomViewer.setParam("serverurl", "http://s7dl.scene7.com/is/image/");
basicZoomViewer.setAsset("Scene7SharedAssets/Backpack_B");
basicZoomViewer.init();
</script>
</body>
</html>
```

Command reference – Configuration attributes

Configuration attributes documentation for Basic Zoom Viewer.

See also [Command reference common to all viewers – Configuration attributes](#)

closebutton

closebutton=0|1

0 1	Set to 1 to enable close button display, or set to 0 to hide close button.
-----	--

Properties

Optional.

Default

0

Example

closebutton=1

ZoomView.doubleclick

[ZoomView.<containerId>_zoomView.]doubleclick=none|zoom|reset|zoomReset

none zoom reset zoomReset	Configures the mapping of double-click/tap to zoom actions. Setting to none disables double-click/tap zoom. If set to zoom clicking the image zooms in one zoom step; CTRL+Click zooms out one zoom step. Setting to reset causes a single click on the image to reset the zoom to the initial zoom level. For zoomReset, reset is applied if the current zoom factor is at or beyond the specified limit, otherwise zoom is applied.
---------------------------	---

Properties

Optional.

Default

reset on desktop computers; zoomReset on touch devices.

Example

doubleclick=zoom

ZoomView.enableHD

[ZoomView.<containerId>_zoomView.]enableHD=always|never|limit[,number]

always never limit	Enable, limit or disable optimization for devices where devicePixelRatio is greater than 1, that is devices with high-density display like iPhone4 and similar devices. If active then the component limits the size of the IS image request as if the device only had a pixel ratio of 1 and that way reducing the bandwidth. See example below.
number	If using the limit setting, the component enables high pixel density only up to the specified limit. See example below.

Properties

Optional.

Default

limit,1500

Example

The following are the expected results when you use this configuration attribute with the viewer, and the viewer size is 1000 x 1000:

If the set variable equals	Result
always	<p>The pixel density of the screen/device is always taken into account.</p> <ul style="list-style-type: none"> • If the screen pixel density = 1, then the requested image is 1000 x 1000. • If the screen pixel density = 1.5, then the requested image is 1500 x 1500. • If the screen pixel density = 2, then the requested image is 2000 x 2000.
never	<p>It always uses a pixel density of 1 and ignores the device's HD capability. Therefore, the requested image requested is always 1000 x 1000.</p>
limit<number>	<p>A device pixel density is requested and served only if the resulting image is below the specified limit.</p> <p>The limit number applies to either the width or the height dimension.</p> <ul style="list-style-type: none"> • If the limit number is 1600, and the pixel density is 1.5, then the 1500 x 1500 image is served. • If the limit number is 1600, and the pixel density is 2, then the 1000 x 1000 image is served because the 2000 x 2000 image exceeds the limit. <p>Best practice: The limit number needs to work in conjunction with the company setting for maximum size image. Therefore, set the limit number to equal the company maximum image size setting.</p>

ZoomView.fmt

[ZoomView.<containerId>_zoomView.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha

jpg jpeg png png-alpha gif gif-alpha	<p>Specifies the image format that the component uses for loading images from Image Server. If the specified format ends with "-alpha", the component renders images as transparent content. For all other image formats the component treats images as opaque. The component has a white background by default. Therefore, to make it transparent, set the background-color CSS property to transparent.</p>
--------------------------------------	---

Properties

Optional.

Default

jpeg

Example

fmt=png-alpha

ZoomView.iconeffect

```
[ZoomView.|<containerId>_zoomView.]iconeffect=0|1[,count][,fade][,autoHide]
```

0 1	Enables the IconEffect to display on the top of the image when the image is in a reset state and it is suggestive of an available action to interact with the image.
<i>count</i>	Specifies the maximum number of times the IconEffect appears and reappears. A value of -1 indicates that the icon always reappears indefinitely.
<i>fade</i>	Specifies the duration of the show or hide animation, in seconds.
<i>autoHide</i>	Sets the number of seconds that the IconEffect stays fully visible before it auto hides. That is, the time after the fade-in animation is completed but before the fade out animation starts. A setting of 0 disables the auto-hide behavior.

Properties

Optional.

Default


1,1,0.3,3

Example

iconeffect=0

ZoomView.iscommand

```
[ZoomView.|<containerId>_zoomView.]iscommand=isCommand
```

<i>iscommand</i>	<p>The Image Serving command string that is applied to zoom image. If specified in the URL all occurrences of & and = must be HTTP-encoded as %26 and %3D, respectively.</p> <p> Note: Image sizing manipulation commands are not supported.</p>
------------------	--

Properties

Optional.

Default

None.

Example

When specified in the viewer URL:

```
iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000
```

When specified in the config data:

```
iscommand=op_sharpen=1&op_colorize=0xff0000
```

ZoomView.reset

```
[ZoomView.|<containerId>_zoomView.]reset=0|1
```

0 1	Resets the view port when frame (image) changes. If set to 0 it preserves the current view port with the best possible fit while preserving the aspect ratio of the newly set image.
-----	--

Properties

Optional.

Default

1

Example

```
reset=0
```

ZoomView.rgn

```
[ZoomView.|<containerId>_zoomView.]rgn=x,y,w,h
```

x,y,w,h	Initial region of interest in pixel coordinates. If these value are not specified, the entire image is fitted within the initial viewport.
---------	--

Properties

Optional.

Default

None.

Example

`rgn=0,0,500,500`

ZoomView.rgnN

`[ZoomView.<containerId>_zoomView.]rgnN=x,y,w,h`

<code>x,y,w,h</code>	Initial region of interest in normalized coordinates. If these values are not specified, entire image is fitted within the initial viewport.
----------------------	--

Properties

Optional.

Default

None.

Example

`rgn=0,0,0.5,0.5`

ZoomView.singleclick

`[ZoomView.<containerId>_zoomView.]singleclick=none|zoom|reset|zoomReset`

<code>none zoom reset zoomReset</code>	Configures the mapping of single-click/tap to zoom actions. Setting to <code>none</code> disables single-click/tap zoom. If set to <code>zoom</code> clicking the image zooms in one zoom step; CTRL+Click zooms out one zoom step. Setting to <code>reset</code> causes a single click on the image to reset the zoom to the initial zoom level. For <code>zoomReset</code> , reset is applied if the current zoom factor is at or beyond the specified limit, otherwise zoom is applied.
--	--

Properties

Optional.

Default

`zoomReset` on desktop computers; `none` on touch devices.

Example

`singleclick=zoom`

ZoomView.transition

`[ZoomView.<containerId>_zoomView.]transition=time[,easing]`

<code>time</code>	Specifies the time in seconds that the animation for a single zoom step action takes.
-------------------	---

<i>easing</i>	<p>Creates an illusion of acceleration or deceleration which makes the transition appear more natural. You can set easing to one of the following:</p> <ul style="list-style-type: none"> • 0 (auto) • 1 (linear) • 2 (quadratic) • 3 (cubic) • 4 (quartic) • 5 (quintic) <p>Auto mode always uses linear transition when elastic zoom is disabled (default). Otherwise, it fits one of the other easing functions based on the transition time. That is, the shorter the transition time the higher the easing function is used to accelerate the acceleration or deceleration effect.</p>
---------------	---

Properties

Optional.

Default

0.5,0

Example

```
transition=2,2
```

ZoomView.zoomstep

```
[ZoomView.<containerId>_zoomView.]zoomstep=step[,limit]
```

<i>step</i>	Configures the number of zoom in and zoom out actions that are required to increase or decrease the resolution by a factor of two. The resolution change for each zoom action is $2^{\pm 1}$ per step. Set to 0 to zoom to full resolution with a single zoom action.
<i>limit</i>	Specifies the maximum zoom resolution, relative to the full resolution image. The default is 1.0, which does not allow zooming beyond full resolution.

Properties

Optional.

Default

1,1

Example

```
zoomstep=2,3
```

Javascript API reference for Basic Zoom Viewer

The main class of the Basic Zoom Viewer is `BasicZoomViewer`. It is declared in the `s7viewers` namespace. This JavaScript API covers constructor, methods, and call backs of this particular class.

In all the following examples, `<instance>` stands for the actual name of the JavaScript viewer object that is instantiated from the `s7viewers.BasicZoomViewer` class.

BasicZoomViewer

JavaScript API reference for Basic Zoom Viewer.

```
BasicZoomViewer([config])
```

Constructor, creates a new Basic Zoom Viewer instance.

Parameters

<i>config</i>	<p><code>{object}</code> optional JSON configuration object, allows all the viewer settings to pass to the constructor to avoid calling individual setter methods. Contains the following properties:</p> <ul style="list-style-type: none"> • <code>containerId</code> – <code>{String}</code> ID of the DOM container (normally a <code>DIV</code>) that the viewer is inserted into. By the time this method is called, it is not necessary to have the container element created. However, the container must exist when <code>init()</code> is run. Required. • <code>params</code> – <code>{Object}</code> JSON object with viewer configuration parameters where the property name is either viewer-specific configuration option or SDK modifier, and the value of that property is a corresponding settings value. Required. • <code>handlers</code> – <code>{Object}</code> JSON object with viewer event callbacks, where the property name is the name of supported viewer event and the property value is a JavaScript function reference to appropriate callback. Optional. <p>See Event callbacks for more information about viewer events.</p> • <code>localizedTexts</code> – <code>{Object}</code> JSON object with localization data. Optional. <p>See Localization of user interface elements for more information.</p> <p>See also the <i>Viewer SDK User Guide</i> and the example for more information about the object's content. Optional</p>
---------------	---

Returns

None.

Example

```
var basicZoomViewer = new s7viewers.BasicZoomViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/Backpack_B",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  },
  "handlers": {
    "initComplete": function() {
      console.log("init complete");
    }
  }
});
```

```
}
},
"localizedTexts": {
  "en": {
    "CloseButton.TOOLTIP": "Close"
  },
  "fr": {
    "CloseButton.TOOLTIP": "Fermer"
  },
  defaultLocale: "en"
}
});
```

dispose

JavaScript API reference for Basic Zoom Viewer.

```
dispose()
```

Disposes this viewer instance by releasing all resources used by the viewer logic and deleting all inner objects and components created by the viewer in runtime.

The web page code should also delete the viewer instance variable as well to completely remove the viewer from the web browser memory.

If the web page code has registered event listeners directly on Viewer SDK components used by the viewer—or stored external references to such components—such listeners must be explicitly unregistered by the web page code, and such external component references must be deleted prior to calling `dispose()`.

Do not access the Viewer API any more after `dispose()` is called.

Parameters

None.

Returns

None.

Example

```
<instance>.dispose()
```

getComponent

JavaScript API reference for Basic Zoom Viewer

```
getComponent(componentId)
```

Returns a reference to the Viewer SDK component that is used by the viewer. The web page can use this method to extend or customize the behavior of the out-of-box viewer. Call this method only after the `initComplete` viewer callback has run, otherwise the component may not be created yet by the viewer logic.

Parameters

componentID – {String} an ID of the Viewer SDK component used by the viewer. This viewer supports the following component IDs:

Component ID	Viewer SDK component class name
parameterManager	s7sdk.ParameterManager
container	s7sdk.common.Container
mediaSet	s7sdk.set.MediaSet
zoomView	s7sdk.image.ZoomView
zoomInButton	s7sdk.common.ZoomInButton
zoomOutButton	s7sdk.common.ZoomOutButton
zoomResetButton	s7sdk.common.ZoomResetButton
fullScreenButton	s7sdk.common.FullScreenButton
closeButton	s7sdk.common.CloseButton

When working with SDK APIs it is important to use correct fully qualified SDK namespace as described in Viewer SDK namespace. Refer to the Viewer SDK API documentation for more information about a particular component.

Returns

{Object} a reference to Viewer SDK component. The method returns null if the componentId is not a supported viewer component or if the component was not yet created by the viewer logic.

Example

```
<instance>.setHandlers({  
  "initComplete":function() {  
    var zoomView = <instance>.getComponent("zoomView");  
  }  
})
```

init

JavaScript API reference for Basic Zoom Viewer.

```
init()
```

Starts the initialization of the Basic Zoom Viewer. By this time the container DOM element must be created so that the viewer code can find it by its ID.

If the container element is not a part of the web page layout just yet (for example, it may be hidden using `display:none` style assigned to it), the viewer suspends its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load resumes automatically.

Call this method only once during the viewer life cycle; subsequent calls are ignored.

Parameters

None.

Returns

{Object} A reference to viewer instance.

Example

```
<instance>.init()
```

setAsset

JavaScript API reference for Basic Zoom Viewer.

setAsset(asset)

asset	{String} new asset id, with optional IS modifiers appended after "?" Images that use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.
-------	--

Sets the new asset. You can call this parameter at any time, either before or after `init()`. If it is called after `init()`, the viewer swaps the asset at runtime.

See also [init](#).

Returns

None.

Example

Single image reference:

```
<instance>.setAsset("Scene7SharedAssets/Backpack_B")
```

Sharpening modifier added to all images in the set:

```
<instance>.setAsset("Scene7SharedAssets/Backpack_B?op_sharpen=1")
```

setContainerId

JavaScript API reference for Basic Zoom Viewer.

setContainerId(containerId)

Sets the ID of the DOM container (normally a DIV) into which the viewer is inserted. It is not necessary to have the container element created by the time this method is called. However, the container must exist when `init()` is run. It must be called before `init()`.

This method is optional if the viewer configuration information was passed with `config` JSON object to constructor.

containerId	{string} ID of container.
-------------	---------------------------

Returns

None.

Example

```
<instance>.setContainerId("s7viewer");
```

setHandlers

JavaScript API reference for Basic Zoom Viewer

```
setHandlers(handlers)
```

Specifies zero or more callback handlers. A call to this method fully overwrites event handlers that were previously assigned for that viewer instance. Must be called before `init()`.

Parameter

<i>handlers</i>	<p>{Object} JSON object with viewer event callbacks, where the property name is the name of the supported viewer event and the property value is a JavaScript function reference to an appropriate callback.</p> <p>See Event callbacks for more information about viewer events.</p>
-----------------	---

Returns

None.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    console.log("init complete");
  }
})
```

setLocalizedTexts

JavaScript API reference for Basic Zoom Viewer.

```
setLocalizedTexts(localizationInfo)
```

<i>localizationInfo</i>	<p>{Object} JSON object with localization data.</p> <p>See Localization of user interface elements for more information.</p> <p>See also the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
-------------------------	--

Sets localization SYMBOL values for one or more locales. This parameter must be called before `init()`.

See also [init](#).

Returns

None.

Example

```
<instance>.setLocalizedTexts({ "en": { "CloseButton.TOOLTIP": "Close" }, "fr": { "CloseButton.TOOLTIP": "Fermer" }, defaultLocale: "en" })
```

setParam

JavaScript API reference for Basic Zoom Viewer.

```
setParam(name, value)
```

Sets the viewer parameter to a specified value. The parameter is either a viewer-specific configuration option or a software development kit modifier. This parameter is called before `init()`.

This method is optional if the viewer configuration information was passed with `config` JSON object to constructor.

See also [init](#).

<i>name</i>	{string} name of parameter.
<i>value</i>	{string} value of parameter. The value cannot be percent-encoded.

Returns

None.

Example

```
<instance>.setParam("style", "customStyle.css")
```

setParams

JavaScript API reference for Basic Zoom Viewer.

```
setParams(params)
```

Sets one or more parameters to a given value. The method argument syntax is identical to a URL query string. That is, it represents name=value pairs separated with `&`. Just as in a query string, the names and values are percent-encoded using UTF8. Before you call `init()`, this parameter must be called.

This method is optional if the viewer configuration information was passed with `config` JSON object to constructor.

See also [init](#).

<i>params</i>	{string} name=value parameter pairs separated with <code>&</code> .
---------------	---

Returns

None.

Example

```
<instance>.setParams("ZoomView.zoomfactor=2,3&ZoomView.iscommand=op_sharpen%3d1")
```


Event callbacks

The viewer supports JavaScript event callbacks that the web page uses to track the viewer initialization process or runtime behavior.

Callback handlers are assigned by passing event names and corresponding handler functions with the `handlers` property to `config` JSON object in the viewer's constructor. Alternatively, it is possible to use `setHandlers()` API method.

Supported viewer events include the following:

- `initComplete` – triggers when viewer initialization is complete and all internal components are created, so that it is possible to use `getComponent()` API. The callback handler does not take any arguments.
- `trackEvent` – triggers each time an event occurs inside the viewer which may be handled by an event tracking system, such as Adobe Analytics. The callback handler takes the following arguments:
 - `objID` {String} not currently used.
 - `compClass` {String} not currently used.
 - `instName` {String} an instance name of the Viewer SDK component that triggered the event.
 - `timeStamp` {Number} event time stamp.
 - `eventInfo` {String} event payload.

See also [BasicZoomViewer](#) and [setHandlers](#).

Customizing Basic Zoom Viewer

All visual customization and most behavior customization for the Basic Zoom Viewer is done by creating a custom CSS.

The suggested workflow is to take the default CSS file for the appropriate viewer, copy it to a different location, customize it, and specify the location of the customized file in the `style=` command.

Default CSS files can be found at the following location:

```
<s7_viewers_root>/html5/BasicZoomViewer_light.css
```

The viewer is supplied with two out-of-the-box CSS files, for "light" and "dark" color schemes. The "light" version is used by default, but you can switch to the "dark" version by using the following standard CSS:

```
<s7_viewers_root>/html5/BasicZoomViewer_dark.css
```

Custom CSS file must contain the same class declarations as the default one. If a class declaration is omitted, the viewer does not function properly because it does not provide built-in default styles for user interface elements.

Alternative way to provide custom CSS rules is to use embedded styles directly on the web page or in one of linked external CSS rules.

When creating custom CSS keep in mind that the viewer assigns `.s7basiczoomviewer` class to its container DOM element. If you are using external CSS file passed with `style=` command, use `.s7basiczoomviewer` class as parent class in descendant selector for your CSS rules. If you are doing embedded styles on the web page, additionally qualify this selector with an ID of the container DOM element as follows:

```
#<containerId>.s7basiczoomviewer
```

Building responsive design CSS

It is possible to target different devices and embedding sizes in CSS to make your content display differently depending on a user's device or a particular web page layout. This includes, but is not limited to, different layouts, user interface element sizes, and artwork resolution.

The viewer supports two mechanisms of creating responsive designed CSS: CSS Markers and standard CSS media queries. You can use these independently or together.

CSS markers

To assist in creating responsive designed CSS, the viewer supports CSS Markers. These are special CSS classes that are dynamically assigned to the top-level viewer container element based on the run-time viewer size and the input type used on the current device.

The first group of CSS markers includes `.s7size_large`, `.s7size_medium`, and `.s7size_small` classes. They are applied based on the run-time area of the viewer container. If the viewer area is equal or bigger than the size of a common desktop monitor then `.s7size_large` is used; if the area is close to a common tablet device then `.s7size_medium` is assigned. For areas similar to mobile phone screens then `.s7size_small` is set. The primary purpose of these CSS markers is to create different user interface layouts for different screens and viewer sizes.

The second group of CSS markers contains `.s7mouseinput` and `.s7touchinput`. `.s7touchinput` is set if the current device has touch input capabilities; otherwise, `.s7mouseinput` is used. These markers are mostly intended to create user interface input elements with different screen sizes for different input types, because touch input normally requires larger elements. In cases where the device has both mouse input and touch capabilities, `.s7touchinput` is set and the viewer renders a touch-friendly user interface.

The following sample CSS sets the zoom in button size to 28 x 28 pixels on systems with mouse input and to 56 x 56 pixels on touch devices. In addition, it hides the button completely if the viewer size gets very small:

```
.s7basiczoomviewer.s7mouseinput .s7zoominbutton {
    width:28px;
    height:28px;
}
.s7basiczoomviewer.s7touchinput .s7zoominbutton {
    width:56px;
    height:56px;
}
.s7basiczoomviewer.s7size_small .s7zoominbutton {
    visibility:hidden;
}
```

To target devices with different pixel density you need to use CSS media queries. The following media query block would contain CSS specific to high-density screens:

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5)
{
}
```

Using CSS markers is the most flexible way of building CSS for responsive design because it lets you target not only the device screen size but the actual viewer size, which is useful for responsive designed layouts.

You can use the default viewer CSS file as an example of a CSS Markers approach.

CSS media queries

You can also accomplish device sensing by using pure CSS Media Queries. Everything enclosed within a given media query block is applied only when it is run on a corresponding device.

When applied to Mobile Viewers use four CSS media queries, defined in your CSS, in the order listed below:

1. Contains only rules specific for all touch devices.

```
@media only screen and (max-device-width:13.5in) and (max-device-
height:13.5in) and (max-device-width:799px),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in)
and (max-device-height:799px)
{
}
```

2. Contains only rules specific for tablets with high resolution screens.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px) and (-webkit-min-device-pixel-ratio:1.5),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px) and (-webkit-min-device-pixel-ratio:1.5)
{
}
```

3. Contains only rules specific for all mobile phones.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in)
{
}
```

4. Contains only rules specific for mobile phones with high resolution screens.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in) and
(-webkit-min-device-pixel-ratio: 1.5),
only screen and (device-width:720px) and (device-height:1280px) and
(-webkit-device-pixel-ratio: 2),
only screen and (device-width:1280px) and (device-height:720px) and
(-webkit-device-pixel-ratio: 2)
{
}
```

Using a media queries approach, you should organize CSS with device sensing as follows:

- First, the desktop-specific section defines all properties that are either desktop-specific or common to all screens.
- And second, the four media queries go in the order defined above and provide CSS rules that are specific for the corresponding device type.

There is no need to duplicate the entire viewer CSS in each media query. Only properties that are specific to given devices are redefined inside a media query.

CSS Sprites

Many viewer user interface elements are styled using bitmap artwork and have more than one distinct visual state. A good example is a button that normally has at least 3 different states: "up", "over", and "down". Each state requires its own bitmap artwork assigned.

With a classic approach to styling, the CSS would have a separate reference to individual image file on the server for each state of the user interface element. The following is a sample CSS for styling the zoom-in button:

```
.s7basiczoomviewer.s7mouseinput .s7zoominbutton[state='up'] {
background-image:url(images/v2/ZoomInButton_dark_up.png);
}
.s7basiczoomviewer.s7mouseinput .s7zoominbutton[state='over'] {
background-image:url(images/v2/ZoomInButton_dark_over.png);
}
.s7basiczoomviewer.s7mouseinput .s7zoominbutton[state='down'] {
background-image:url(images/v2/ZoomInButton_dark_down.png);
}
.s7basiczoomviewer.s7mouseinput .s7zoominbutton[state='disabled'] {
```

```
background-image:url(images/v2/ZoomInButton_dark_disabled.png);
}
```

The drawback to this approach is that the end user experiences flickering or delayed user interface response when the element is interacted with for the first time. This action occurs because the image artwork for the new element state is not yet downloaded. Also, this approach may have a slight negative impact on performance because of an increase in the number of HTTP calls to the server.

CSS sprites is a different approach where image artwork for all element states is combined into a single PNG file called a "sprite". Such "sprite" has all visual states for the given element positioned one after another. When styling a user interface element with sprites the same sprite image is referenced for all different states in the CSS. Also, the `background-position` property is used for each state to specify which part of the "sprite" image is used. You can structure a "sprite" image in any suitable way. Viewers normally have it vertically stacked. Below is a "sprite"-based example of styling the same zoom-in button from above:

```
.s7basiczoomviewer .s7zoominbutton[state] {
    background-image: url(images/v2/ZoomInButton_dark_sprite.png);
}
.s7basiczoomviewer.s7mouseinput .s7zoominbutton[state='up'] {
background-position: -84px -560px;
}
.s7basiczoomviewer.s7mouseinput .s7zoominbutton[state='over'] {
background-position: -56px -560px;
}
.s7basiczoomviewer.s7mouseinput .s7zoominbutton[state='down'] {
background-position: -28px -560px;
}
.s7basiczoomviewer.s7mouseinput .s7zoominbutton[state='disabled'] {
background-position: -0px -560px;
}
```

General styling notes and advice

- When customizing the viewer user interface with CSS the use of the `!important` rule is not supported to style viewer elements. In particular, `!important` rule should not be used to override any default or run-time styling provided by the viewer or Viewer SDK. The reason is that it may affect the behavior of proper components. Instead, you should use CSS selectors with the proper specificity to set CSS properties that are documented in this reference guide.
- All paths to external assets within CSS are resolved against the CSS location, not the viewer HTML page location. Be aware of this rule when you copy the default CSS to a different location. Either copy the default assets as well or update paths within the custom CSS.
- The preferred format for bitmap artwork is PNG.
- Bitmap artwork is assigned to user interface elements using the `background-image` property.
- The width and height properties of a user interface element define its logical size. The size of the bitmap passed to `background-image` does not affect logical size.
- To use the high pixel density of high-resolution screens like Retina, specify bitmap artwork twice as large as the logical user interface element size. Then, apply the `-webkit-background-size:contain` property to scale the background down to the logical user interface element size.
- To remove a button from the user interface, add `display:none` to its CSS class.
- You can use various formats for color value that CSS supports. If you need transparency, use the format `rgba(R,G,B,A)`. Otherwise, you can use the format `#RRGGBB`.

Common User Interface Elements

The following is user interface elements reference documentation that applies to Basic Zoom Viewer:

Close button

Clicking or tapping this button closes the containing web page. This button only appears if the `closebutton` parameter is set to 1. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7basiczoomviewer .s7closebutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a close button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7basiczoomviewer .s7closebutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7basiczoomviewer .s7closebutton [state='up'] {
background-image:url(images/v2/CloseButton_dark_up.png);
}
.s7basiczoomviewer .s7closebutton [state='over'] {
background-image:url(images/v2/CloseButton_dark_over.png);
}
.s7basiczoomviewer .s7closebutton [state='down'] {
background-image:url(images/v2/CloseButton_dark_down.png);
}
.s7basiczoomviewer .s7closebutton [state='disabled'] {
```

```
background-image:url(images/v2/CloseButton_dark_disabled.png);
}
```

Focus highlight

Input focus highlight displayed around focused viewer UI element is controlled with the CSS class selector.

CSS properties

The appearance is controlled with the following CSS class selector:

```
.s7basiczoomviewer *:focus
```

CSS property	Description
outline	Focus highlight style.

Example – to disable the default browser focus highlight for all viewer user interface elements, add the following CSS selector to viewer's style sheet:

```
.s7basiczoomviewer *:focus {
  outline: none;
}
```

Full screen button

Causes the viewer to enter or exit full screen mode when clicked by the user. This button does not display if the viewer works in pop-up mode and the system does not support native full screen. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7basiczoomviewer .s7fullscreenbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used.

CSS property	Description
	See CSS Sprites .



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the "full screen" state and *selected='false'* corresponds to the "normal" state.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a full screen button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states when selected or not selected:

```
.s7basiczoomviewer .s7fullscreenbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7basiczoomviewer .s7fullscreenbutton [selected='false'][state='up'] {
background-image:url(images/enterFullBtn_up.png);
}
.s7basiczoomviewer .s7fullscreenbutton [selected='false'][state='over'] {
background-image:url(images/enterFullBtn_over.png);
}
.s7basiczoomviewer .s7fullscreenbutton [selected='false'][state='down'] {
background-image:url(images/enterFullBtn_down.png);
}
.s7basiczoomviewer .s7fullscreenbutton [selected='false'][state='disabled'] {
background-image:url(images/enterFullBtn_disabled.png);
}
.s7basiczoomviewer .s7fullscreenbutton [selected='true'][state='up'] {
background-image:url(images/exitFullBtn_up.png);
}
.s7basiczoomviewer .s7fullscreenbutton [selected='true'][state='over'] {
background-image:url(images/exitFullBtn_over.png);
}
.s7basiczoomviewer .s7fullscreenbutton [selected='true'][state='down'] {
background-image:url(images/exitFullBtn_down.png);
}
.s7basiczoomviewer .s7fullscreenbutton [selected='true'][state='disabled'] {
background-image:url(images/exitFullBtn_disabled.png);
}
```

Icon effect

The zoom indicator is overlaid on the main view area. It is displayed when the image is in a reset state and it also depends on *iconeffect* parameter.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7basiczoomviewer .s7zoomview .s7iconeffect
```

CSS property	Description
background-image	Zoom indicator artwork.

CSS property	Description
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
width	Zoom indicator width.
height	Zoom indicator height.



Note: Icon effect supports *media-type* attribute selector, which you can use to apply different icon effects on different devices. In particular, *media-type='standard'* corresponds to desktop systems where mouse input is normally used and *media-type='multitouch'* corresponds to devices with touch input.

Example – to set up a 100 x 100 pixel zoom indicator with different art for desktop systems and touch devices.

```
.s7basiczoomviewer .s7zoomview .s7iconeffect {
  width: 100px;
  height: 100px;
}
.s7basiczoomviewer .s7zoomview .s7iconeffect[media-type='standard'] {
  background-image:url(images/v2/IconEffect_zoom.png);
}
.s7basiczoomviewer .s7zoomview .s7iconeffect[media-type='multitouch'] {
  background-image:url(images/v2/IconEffect_pinch.png);
}
```

Main viewer area

The main view area is the area occupied by the zoom image. It usually sets to fit the available device screen when no size is specified.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7basiczoomviewer
```

CSS property	Description
width	The width of the viewer.
height	The height of the viewer.
background-color	Background color in hexadecimal format.

Example – to set up a viewer with white background (#FFFFFF) and make its size 512 x 288 pixels.

```
.s7basiczoomviewer{
  background-color: #FFFFFF;
  width: 512px;
  height: 288px;
}
```


Tooltips

On desktop systems some user interface elements like buttons have tooltips that are displayed on mouse hover.

CSS properties of the main viewer area

The appearance of tooltips is controlled with the following CSS class selector:

```
.s7tooltip
```

CSS property	Description
border-radius	Background border radius.
border-color	Background border color.
background-color	Background color.
color	Text color.
font-family	Text font name.
font-size	Text font size.



Note: In case tooltip styles are customized from within the embedding web page, all properties have to contain **! IMPORTANT** rule. This is not necessary if tooltips are customized in the viewer's CSS file.

Example – to set up tooltips that have a grey border with 3px corner radius, black background and white text written with Arial, 11 pixels size:

```
.s7tooltip {
border-radius: 3px 3px 3px 3px;
border-color: #999999;
background-color: #000000;
color: #FFFFFF;
font-family: Arial, Helvetica, sans-serif;
font-size: 11px;
}
```

Zoom in button

Clicking or tapping this button zooms in on an image in the main view. This button does not display on mobile phones in order to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7basiczoomviewer .s7zoominbutton
```

CSS property	Description
top	Position from the top border, including padding.

CSS property	Description
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom in button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7basiczoomviewer .s7zoominbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7basiczoomviewer .s7zoominbutton [state='up'] {
background-image:url(images/v2/ZoomInButton_dark_up.png);
}
.s7basiczoomviewer .s7zoominbutton [state='over'] {
background-image:url(images/v2/ZoomInButton_dark_over.png);
}
.s7basiczoomviewer .s7zoominbutton [state='down'] {
background-image:url(images/v2/ZoomInButton_dark_down.png);
}
.s7basiczoomviewer .s7zoominbutton [state='disabled'] {
background-image:url(images/v2/ZoomInButton_dark_disabled.png);
}
```

Zoom out button

Clicking or tapping this button zooms out on an image in the main view. This button does not display on mobile phones in order to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7basiczoomviewer .s7zoomoutbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom out button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7basiczoomviewer .s7zoomoutbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7basiczoomviewer .s7zoomoutbutton [state='up'] {
background-image:url(images/v2/ZoomOutButton_dark_up.png);
}
.s7basiczoomviewer .s7zoomoutbutton [state='over'] {
background-image:url(images/v2/ZoomOutButton_dark_over.png);
}
.s7basiczoomviewer .s7zoomoutbutton [state='down'] {
background-image:url(images/v2/ZoomOutButton_dark_down.png);
}
.s7basiczoomviewer .s7zoomoutbutton [state='disabled'] {
background-image:url(images/v2/ZoomOutButton_dark_disabled.png);
}
```

Zoom reset button

Clicking or tapping this button resets an image in the main view. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7basiczoomviewer .s7zoomresetbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom reset button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7basiczoomviewer .s7zoomresetbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7basiczoomviewer .s7zoomresetbutton [state='up'] {
background-image:url(images/v2/ZoomResetButton_dark_up.png);
}
.s7basiczoomviewer .s7zoomresetbutton [state='over'] {
background-image:url(images/v2/ZoomResetButton_dark_over.png);
}
.s7basiczoomviewer .s7zoomresetbutton [state='down'] {
background-image:url(images/v2/ZoomResetButton_dark_down.png);
}
.s7basiczoomviewer .s7zoomresetbutton [state='disabled'] {
background-image:url(images/v2/ZoomResetButton_dark_disabled.png);
}
```

Zoom view

Main view consists of the zoomable image.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7basiczoomviewer .s7zoomview
```

CSS property	Description
background-color	Background color in hexadecimal format of the main view.
cursor	Cursor is displayed over the main view.

Example – to make the main view transparent.

```
.s7basiczoomviewer .s7zoomview {
  background-color: transparent;
}
```

On desktop systems the component supports the `cursorType` attribute selector which can be applied to the `.s7zoomview` class and controls the cursor type based on the component state and user action. The following `cursorType` values are supported:

Value	Description
default	Displayed when the image is not zoomable because of a small image resolution, or component settings, or both.
zoomin	Displayed when the image can be zoomed in on.
reset	Displayed when the image is at maximum zoom level and can be reset to its initial state.
drag	Displayed when a user pans the image which is in zoomed in state.

Support for Adobe Analytics tracking

The Basic Zoom Viewer supports Adobe Analytics tracking out of the box.

Out-of-the-box tracking

The Basic Zoom Viewer supports Adobe Analytics tracking out-of-the-box. To enable tracking, pass the proper company preset name as `config2` parameter.

The viewer also sends a single tracking HTTP request to the configured Image Server with the viewer type and version information.

Custom tracking

To integrate with third-party analytics systems it is necessary to listen to the `trackEvent` viewer callback and process the `eventInfo` argument of the callback function as necessary. The following code is an example of such handler function:

```
var basicZoomViewer = new s7viewers.BasicZoomViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/Backpack_B",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  },
  "handlers": {
    "trackEvent": function(objID, compClass, instName, timeStamp, eventInfo) {
      //identify event type
      var eventType = eventInfo.split(",")[0];
      switch (eventType) {
```

```

    case "LOAD":
        //custom event processing code
        break;
    //additional cases for other events
}
}
}
);

```

The viewer tracks the following SDK user events:

SDK user event	Sent when...
LOAD	viewer is loaded first.
SWAP	an asset is swapped in the viewer using the <code>setAsset ()</code> API.
ZOOM	an image is zoomed.
PAN	an image is panned.

Localization of user interface elements

Certain content that the Basic Zoom Viewer displays is subject to localization, including zoom buttons and a full screen button.

Every textual content in the viewer that can be localized is represented by a special Viewer SDK identifier called SYMBOL. Any SYMBOL has a default associated text value for the English locale ("en") supplied with the out-of-the-box viewer, and also may have user-defined values set for as many locales as needed.

When the viewer starts, it checks the current locale to see if there is a user-defined value for each supported SYMBOL in the locale. If there is, it uses the user-defined value; otherwise, it falls back to the out-of-the-box default text.

User-defined localization data can be passed to the viewer as a localization JSON object. Such an object contains the list of supported locales, SYMBOL text values for each locale, and the default locale.

An example of such localization object:

```

{
  "en": {
    "CloseButton.TOOLTIP": "Close",
    "ZoomInButton.TOOLTIP": "Zoom In"
  },
  "fr": {
    "CloseButton.TOOLTIP": "Fermer",
    "ZoomInButton.TOOLTIP": "Agrandir"
  },
  defaultLocale: "en"
}

```

In the example above, the localization object defines two locales ("en" and "fr") and provides localization for two user interface elements in each locale.

The web page code should pass such localization object to the viewer constructor as a value of `localizedTexts` field of the configuration object. An alternative option is to pass the localization object by calling `setLocalizedTexts(localizationInfo)` method.

The following SYMBOLs are supported:

SYMBOL	Tooltip for the...
<code>CloseButton.TOOLTIP</code>	Close button.
<code>ZoomInButton.TOOLTIP</code>	Zoom in button.
<code>ZoomOutButton.TOOLTIP</code>	Zoom out button.
<code>ZoomResetButton.TOOLTIP</code>	Zoom reset button.
<code>FullScreenButton.TOOLTIP_SELECTED</code>	Full screen button in normal state.
<code>FullScreenButton.TOOLTIP_UNSELECTED</code>	Full screen button in full screen state.

Full Screen Support

The viewer supports full screen operation mode.

On modern desktop browsers, except Internet Explorer 10 and older, and on some touch devices, the viewer uses "native" full screen mode. This mode means that the entire device screen is occupied by the viewer content.

On iOS devices and on older Internet Explorer browsers, the viewer uses "simulated" full screen mode instead. In this mode, the viewer simply resizes to take the full area of the web browser window. Also, the web browser Chrome and other windows are still visible on the screen.

An end user enters and leaves full screen mode by pressing the Full Screen button in the viewer user interface. When "native" full screen mode is used on desktop, it is also possible to exit it by pressing **Esc**.

Viewer SDK namespace

The viewer is built of many Viewer SDK components. In most cases, the web page does not need to interact with SDK components API directly; all common needs are covered in the viewer API itself.

However, some advanced use cases require that the web page obtain a reference to an inner SDK component using the `getComponent()` viewer API and then use all the flexibility of the APIs of SDK itself.

The namespace that is used to load and initialize SDK components by the viewer depends on the environment in which the viewer is operating. If the viewer is running in AEM (Adobe Experience Manager), the viewer loads SDK components into `s7viewers.s7sdk` namespace. And the viewer served from Scene7 Publishing System loads the SDK into `s7classic.s7sdk`.

In either case, the namespace used by the SDK inside the viewer has either `s7viewers` or `s7classic` as the prefix. And, it is different from the plain `s7sdk` namespace used in the SDK User Guide or SDK API documentation.

For that reason it is important to use a fully qualified SDK namespace when you write custom application code that communicates with internal viewer components.

For example, if you plan to listen to the `StatusEvent.NOTF_VIEW_READY` event and the viewer is served by AEM, the fully qualified event type is `s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY`, and the event listener code looks similar to the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var zoomView = <instance>.getComponent("zoomView");
    zoomView.addEventListener(s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});
```

The same code for viewer served from Scene7 SPS will look like this:

```
<instance>.setHandlers({
  "initComplete":function() {
    var zoomView = <instance>.getComponent("zoomView");
    zoomView.addEventListener(s7classic.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});
```

eCatalog

eCatalog Viewer is a catalog viewer that displays electronic brochures in a spread by spread or page by page manner. The eCatalog lets users navigate through the catalog using additional user interface elements or dedicated thumbnails mode. Users can also zoom in on every page for greater detail.



Note: Images which use Image Rendering (IR) or User-Generated Content (UGC) are not supported by this viewer.

Viewer type 507.

See [System requirements](#).

This viewer works with ecatalogs and supports optional image maps and social sharing tools. It has zoom tools, catalog navigation tools, full screen support, thumbnails, and an optional close button. The viewer also supports social sharing tools, Print, Download, and Favorites. It is designed to work on desktops and mobile devices.

Demo URL

<http://s7d1.scene7.com/s7viewers/html5/eCatalogViewer.html?asset=Viewers/Pluralist>

Using eCatalog Viewer

eCatalog Viewer represents a main JavaScript file and a set of helper files (a single JavaScript include with all Viewer SDK components used by this particular viewer, assets, CSS) downloaded by the viewer in runtime

You can use eCatalog Viewer in pop-up mode using a production-ready HTML page provided with IS-Viewers or in embedded mode, where it is integrated into target web page using documented API.

Configuration and skinning are similar to that of the other viewers. All skinning is achieved by way of custom CSS.

See [Command reference common to all viewers – Configuration attributes](#) and [Command reference common to all Viewers – URL](#)

Interacting with eCatalog Viewer

eCatalog Viewer supports the following touch gestures that are common in other mobile applications.

Gesture	Description
Single tap	Selects new thumbnail in swatches.
Double tap	Zooms in one level until maximum magnification is reached. The next double tap gesture resets the viewer to the initial viewing state.
Pinch	Zooms in or out.
Horizontal swipe or flick	Scrolls through the list of catalog pages if a slide frame transition is used.
Vertical swipe or flick	When the image is in a reset state, it performs a native page scroll. When thumbnails are active, it scrolls the list of thumbnails.

It is possible to enable a realistic page flip animation effect for navigating between catalog pages. In such cases, a user can hold and drag a page corner and flip the page.

This viewer also supports both touch input and mouse input on Windows devices with a touch screen and mouse. This support, however, is limited to Chrome, Internet Explorer 11, and Edge web browsers only.

Social media sharing tools with eCatalog Viewer

The eCatalog Viewer supports social sharing tools. They are available as a button in the main control bar which expands into a sharing tool bar when a user clicks or taps on it.

The sharing tool bar contains icons for each type of sharing channel supported which includes Facebook, Twitter, email share, embed code share, and link share. When email share, embed share, or link share tools are activated, the viewer displays a modal dialog box with a corresponding data entry form. When Facebook or Twitter is called, the viewer redirects the user to a standard sharing dialog from a social service. Sharing tools are not available in full screen mode because of web browser security restrictions.

Embedding eCatalog Viewer

Different web pages have different needs for viewer behavior. Sometimes a web page provides a link that, when clicked, opens the viewer in a separate browser window. In other cases, it is necessary to embed the viewer right in the hosting page. In the latter case, the web page may have a static page layout, or use responsive design that displays differently on different devices or for different browser window sizes. To accommodate these needs, the viewer supports three primary operation modes: pop-up, fixed size embedding, and responsive design embedding.

About pop-up mode

In pop-up mode, the viewer is opened in a separate web browser window or tab. It takes the entire browser window area and adjusts in case the browser is resized, or a mobile device's orientation is changed.

Pop-up mode is the most common for mobile devices. The web page loads the viewer using `window.open()` JavaScript call, properly configured A HTML element, or any other suitable method.

It is recommended that you use an out-of-the-box HTML page for pop-up operation mode. In this case, it is called `eCatalogViewer.html` and is located within the `html5/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/eCatalogViewer.html
```

You can achieve visual customization by applying custom CSS.

The following is an example of HTML code that opens the viewer in a new window:

```
<a href="http://s7d1.scene7.com/s7viewers/html5/eCatalogViewer.html?asset=Viewers/Pluralist"
target="_blank">Open popup viewer</a>
```

About fixed size embedding mode and responsive design embedding mode

In the embedded mode, the viewer is added to the existing web page, which may already have some customer content not related to the viewer. The viewer normally occupies only a part of a web page's real estate.

The primary use cases are web pages oriented for desktops or tablet devices, and also responsive designed pages that adjust layout automatically depending on the device type.

Fixed size embedding is used when the viewer does not change its size after initial load. This is the best choice for web pages that have a static layout.

Responsive design embedding assumes that the viewer may need to resize at runtime in response to the size change of its container DIV. The most common use case is adding a viewer to a web page that uses a flexible page layout.

In responsive design embedding mode, the viewer behaves differently depending on the way web page sizes its container DIV. If the web page sets only the width of the container DIV, leaving its height unrestricted, the viewer automatically chooses its height according to the aspect ratio of the asset that is used. This functionality ensures that the asset fits perfectly into the view without any padding on the sides. This use case is the most common for web pages using responsive layout frameworks like Bootstrap, Foundation, and so on.

Otherwise, if the web page sets both the width and the height for the viewer's container DIV, the viewer fills just that area and follows the size that the web page layout provides. A good example is embedding the viewer into a modal overlay, where the overlay is sized according to web browser window size.

Fixed size embedding

You add the viewer to a web page by doing the following:

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Setting the viewer size.
4. Creating and initializing the viewer.

1. Adding the viewer JavaScript file to your web page.

Creating a viewer requires that you add a script tag in the HTML head. Before you can use the viewer API, be sure that you include `eCatalogViewer.js`. The `eCatalogViewer.js` file is located under the `html5/js/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/js/eCatalogViewer.js
```

You can use a relative path if the viewer is deployed on one of the Adobe Scene7 servers and it is served from the same domain. Otherwise, you specify a full path to one of Adobe Scene7 servers that have the IS-Viewers installed.

The relative path looks like the following:

```
<script language="javascript" type="text/javascript"
src="/s7viewers/html5/js/eCatalogViewer.js"></script>
```



Note: You should only reference the main viewer JavaScript `include` file on your page. You should not reference any additional JavaScript files in the web page code which might be downloaded by the viewer's logic in runtime. In particular, do not directly reference HTML5 SDK `Utils.js` library loaded by the viewer from `/s7viewers` context path (so-called consolidated SDK `include`). The reason is that the location of `Utils.js` or similar runtime viewer libraries is fully managed by the viewer's logic and the location changes between viewer releases. Adobe does not keep older versions of secondary viewer `includes` on the server.

As a result, putting a direct reference to any secondary JavaScript `include` used by the viewer on the page breaks the viewer functionality in the future when a new product version is deployed.

2. Defining the container DIV.

Add an empty DIV element to the page where you want the viewer to appear. The DIV element must have its ID defined because this ID is passed later to the viewer API.

The placeholder DIV is a positioned element, meaning that the `position` CSS property is set to `relative` or `absolute`.

The following is an example of a defined placeholder DIV element:

```
<div id="s7viewer" style="position:relative"></div>
```

3. Setting the viewer size

You can set the static size for the viewer by either declaring it for `.s7ecatalogviewer` top-level CSS class in absolute units, or by using `stagesize` modifier.

You can put sizing in CSS directly on the HTML page, or in a custom viewer CSS file, which is then later assigned to a viewer preset record in Scene7 Publishing System, or passed explicitly using a style command.

See [Customizing eCatalog Viewer](#) for more information about styling the viewer with CSS.

The following is an example of defining a static viewer size in HTML page:

```
#s7viewer.s7ecatalogviewer {
  width: 640px;
  height: 480px;
}
```

You can set the `stagesize` modifier either in the viewer preset record in Scene7 Publishing System, or pass it explicitly with the viewer initialization code with `params` collection, or as an API call as described in the Command Reference section, like the following:

```
eCatalogViewer.setParam("stagesize",
  "640,480");
```

4. Initializing the viewer.

When you have completed the steps above, you create an instance of `s7viewers.eCatalogViewer` class, pass all configuration information to its constructor and call `init()` method on a viewer instance. Configuration information is passed to the constructor as a JSON object. At minimum, this object has the `containerId` field which holds the name of viewer container ID and nested `params` JSON object with configuration parameters supported by the viewer. In this case, the `params` object must have at least the Image Serving URL passed as `serverUrl` property, and the initial asset as `asset` parameter. JSON-based initialization API lets you create and start the viewer with single line of code.

It is important to have the viewer container added to the DOM so that the viewer code can find the container element by its ID. Some browsers delay building DOM until the end of the web page. For maximum compatibility, however, call the `init()` method just before the closing `BODY` tag, or on the `body onload()` event.

At the same the container element should not necessarily be part of the web page layout just yet. For example, it may be hidden using `display:none` style assigned to it. In this case, the viewer delays its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load automatically resumes.

The following is an example of creating a viewer instance, passing the minimum necessary configuration options to the constructor, and calling the `init()` method. The example assumes `eCatalogViewer` is the viewer instance; `s7viewer` is the name of placeholder DIV; `http://s7dl.scene7.com/is/image/` is the Image Serving URL, and `Viewers/Pluralist` is the asset:

```
<script type="text/javascript">
var eCatalogViewer = new s7viewers.eCatalogViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Viewers/Pluralist",
    "serverurl":"http://s7dl.scene7.com/is/image/"
  }
}).init();
</script>
```

The following code is a complete example of a trivial web page that embeds the eCatalog Viewer with a fixed size:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/eCatalogViewer.js"></script>
<style type="text/css">
#s7viewer.s7ecatalogviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var eCatalogViewer = new s7viewers.eCatalogViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Viewers/Pluralist",
    "serverurl":"http://s7dl.scene7.com/is/image/"
  }
}).init();
</script>
</body>
</html>
```

Responsive design embedding with unrestricted height

With responsive design embedding, the web page normally has some kind of flexible layout in place that dictates the runtime size of the viewer's container DIV. For purposes of this example, assume that the web page allows the viewer's container DIV to take 40% of the web browser window size, leaving its height unrestricted. The resulting web page HTML code looks like the following:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
.holder {
  width: 40%;
}
</style>
</head>
```

```
<body>
<div class="holder"></div>
</body>
</html>
```

Adding the viewer to such a page is similar to fixed size embedding, with the only difference being that you do not need to explicitly define viewer size.

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Creating and initializing the viewer.

All the steps above are the same as with fixed size embedding. Add the container DIV to the existing "holder" DIV. The following code is a complete example. You can see how the viewer size changes when the browser is resized, and how the viewer aspect ratio matches the asset.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/eCatalogViewer.js"></script>
<style type="text/css">
.holder {
width: 40%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var eCatalogViewer = new s7viewers.eCatalogViewer({
"containerId":"s7viewer",
"params":{
"asset":"Viewers/Pluralist",
"serverurl":"http://s7dl.scene7.com/is/image/"
}
}).init();
</script>
</body>
</html>
```

The following examples page illustrates more real-life use cases of responsive design embedding with unrestricted height:

https://marketing.adobe.com/resources/help/en_US/s7/vlist/vlist.html

Flexible size embedding with width and height defined

In case of flexible-size embedding with width and height defined, the web page styling is different. That is, it provides both sizes to the "holder" DIV and centers it in the browser window. Also, the web page sets the size of the HTML and BODY element to 100%:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
width: 100%;
height: 100%;
}
.holder {
position: absolute;
left: 20%;
top: 20%;
```

```

    width: 60%;
height: 60%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>

```

The remaining embedding steps are identical to responsive design embedding with unrestricted height. The resulting example is the following:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/eCatalogViewer.js"></script>
<style type="text/css">
html, body {
    width: 100%;
    height: 100%;
}
.holder {
    position: absolute;
    left: 20%;
    top: 20%;
    width: 60%;
height: 60%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var eCatalogViewer = new s7viewers.eCatalogViewer({
    "containerId":"s7viewer",
    "params":{
        "asset":"Viewers/Pluralist",
        "serverurl":"http://s7dl.scene7.com/is/image/"
    }
}).init();
</script>
</body>
</html>

```

Embedding Using Setter-based API

Instead of using JSON-based initialization it is possible to use setter-based API and no-args constructor. With that API constructor does not take any parameters and configuration parameters is specified using `setContainerId()`, `setParam()`, and `setAsset()` API methods with separate JavaScript calls.

The following example shows fixed size embedding with setter-based API:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/eCatalogViewer.js"></script>
<style type="text/css">
#s7viewer.s7ecatalogviewer {
    width: 640px;
    height: 480px;
}
</style>
</head>

```

```

<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var eCatalogViewer = new s7viewers.eCatalogViewer();
eCatalogViewer.setContainerId("s7viewer");
eCatalogViewer.setParam("serverurl", "http://s7d1.scene7.com/is/image/");
eCatalogViewer.setAsset("Viewers/Pluralist");
eCatalogViewer.init();
</script>
</body>
</html>

```

Command reference – Configuration attributes

Configuration attributes documentation for eCatalog Viewer.

Any configuration command can be set in URL or using `setParam()`, or `setParams()`, or both, API methods. You can also specify any configuration attribute that is specified in the server-side configuration record.

For some configuration commands you can prefix them with the class name or instance name of corresponding Viewer SDK component. An instance name of the component is dynamic and depends on the ID of the viewer container DOM element passed to `setContainerId()` API method. Documentation includes optional prefix for such commands. For example, `zoomstep` command is documented as follows:

```
[PageView.|<containerId>_pageView].zoomstep
```

which means that you can use this command as:

- `zoomstep` (short syntax)
- `PageView.zoomstep` (qualified with component class name)
- `cont_pageView.zoomstep` (qualified with component ID, assuming `cont` is the ID of the container element)

See also [Command reference common to all viewers – Configuration attributes](#)

Closebutton

```
closebutton=0|1
```

0-1	<p>Set to 1 to enable display of the close button. Or, set to 0 to hide the close button.</p> <p>The close button is supported only on touch devices; it cannot be displayed on desktop systems.</p>
-----	--

Properties

Optional.

Default

0

Example

```
closebutton=1
```

ControlBar.transition

```
[ControlBar.<containerId>_controls.]transition=none|fade[,delaytohide[,duration]
```

<code>none fade</code>	Specifies the effect type that is used to show or hide the control bar and its content. Use <code>none</code> for instant show and hide; <code>fade</code> provides a gradual fade in and fade out effect (not supported on Internet Explorer 8).
<code>delaytohide</code>	Specifies the time in seconds between the last mouse/touch event that the control bar registers and the time control bar hides. If set to <code>-1</code> the component never triggers its auto-hide effect and always stay visible on the screen.
<code>duration</code>	Sets the duration of the fade in and fade out animation, in seconds.

Properties

Optional. This command is ignored on touch devices, where the control bar auto-hide is disabled.

Default

```
fade,2,0.5
```

Example

```
transition=none
```

direction

```
direction=auto|left|right
```

<code>auto left right</code>	<p>Specifies the way pages are displayed in the main view and thumbnails. It also specifies the way the user interacts with the viewer user interface to change between catalog frames.</p> <p>When <code>left</code> is used it sets a right alignment for the initial page, and left alignment for the last page. It stitches individual page sub-images for left-to-right rendering order. It also sets the main view to use right-to-left slide animation to advance the catalog (in case <code>PageView.frametransition</code> is set to <code>slide</code>). Finally, thumbnails are set for a left-to-right fill order.</p> <p>Similarly, when <code>right</code> is used it sets a left alignment for the initial page, and right alignment for the last page. It stitches individual page sub-images for right-to-left rendering order. It also sets the main view to use left-to-right slide animation to advance the catalog (in case <code>PageView.frametransition</code> is set to <code>slide</code>). Finally, it reverses the thumbnails order so that the thumbnails view is filled in right-to-left, top-to-bottom direction.</p> <p>When <code>auto</code> is set, the viewer applies <code>right</code> mode when locale is set to <code>ja</code>; otherwise, it uses <code>left</code> mode.</p>
----------------------------------	---

Properties

Optional.

Default

auto

Example

direction=right

EmailShare.emailurl

[EmailShare.|<containerId>_emailShare.]emailurl=*emailurl*

<i>emailurl</i>	Specifies the base URL for Scene7 OnDemand email service.
-----------------	---

Properties

Optional.

Default

/s7/emailFriend

Example

emailurl=http://s7d1.scene7.com/s7/emailFriend

EmbedShare.embedsizes

[EmbedShare.|<containerId>_embedShare.]embedsizes=*width,height[,0|1][;width,height[,0|1]]*

<i>width</i>	Embed width.
<i>height</i>	Embed height.
0 1	Specifies whether this list item should be initially preselected in the combo box.

Properties

Optional.

Default

1280,960;640,480;320,240

Example

embedsizes=800,600;640,480,1

FavoritesEffect.expiration

[FavoritesEffect. | <containerId>_favoritesEffect.]expiration=days

days	Number of days the collection of favorites is kept on the client's system before they expire. Every time a user visits the catalog and makes a change to the favorites, such as adding or removing, the expiration timer is reset.
------	--

Properties

Optional.

Default

30

Example

expiration=7

FavoritesMenu.bearing

Specifies the direction of slide animation for the buttons container.

[FavoritesMenu. | <containerId>_favoritesMenu.]bearing=up|down|left|right|fit-vertical|fit-lateral

up down left right fit-vertical fit-lateral	<p>When set to up, down, left, or right, the panel rolls out in specified direction without an additional bounds check, which results in panel clipping by an outside container.</p> <p>When set to fit-vertical, the component first shifts the base panel position to the bottom of the Favorites menu and tries to roll out the panel in one of the following directions from such base location: bottom, right, left. With each attempt the component checks if the panel is clipped by an outside container. If all attempts fail, the component tries to shift the base panel position to the top and repeat roll out attempts from a top, right, and left direction.</p> <p>When set to fit-lateral, the component uses a similar logic. The base is shifted to the right first, trying right, down, and up roll out directions. Then, it shifts the base to the left, trying left, down and up roll out directions.</p>
---	---

Properties

Optional.

Default

fit-vertical

Example

bearing=left

FavoritesView.align

[FavoritesView. | <containerId>_favoritesView.]align=left | center | right ,top | center | bottom

left center right ,top center bottom	<p>Specifies the internal horizontal alignment—or anchoring—of the thumbnails container within the component area.</p> <p>In FavoritesView, the internal thumbnail container is sized so that only a whole number of thumbnails are shown. As a result, there is some padding between the internal container and the external component bounds.</p> <p>This modifier specifies how the internal thumbnails container is positioned horizontally inside the component.</p>
--	---

Properties

Optional.

Default

center ,center

Example

align=left ,top

FavoritesView.favoritesThumbView

[FavoritesView. | <containerId>_favoritesView.]favoritesThumbArea=area

area	<p>Specifies the crop area for Favorites thumbnail. Expressed as a relative value to the total frame size, with a range from 0 to 1 . 0.</p> <p>A value of 1 means that the entire frame image is used for the thumbnail.</p> <p>A value of 0 . 1 means only 10% of the frame size is used.</p>
------	---

Properties

Optional.

Default

0 . 1

Example

favoritesThumbArea=0 . 5

FavoritesView.fmt

[FavoritesView. | <containerId>_favoritesView.]fmt=jpg | jpeg | png | png-alpha | gif | gif-alpha

<code>jpg jpeg png png-alpha gif gif-alpha</code>	<p>Specifies the image format used by the component for loading images from Image Server. The format is any value that is supported by the Image Server and the client browser.</p> <p>If the image format ends with <code>-alpha</code>, the component renders the images as transparent content. For all other image format values, the component treats images as opaque.</p>
---	--

Properties

Optional.

Default

jpeg

Example

`fmt=png-alpha`

FavoritesView.iscommand

The Image Serving command string that is applied to all thumbnails.

[FavoritesView. | <containerId>_favoritesView.]iscommand=*isCommand*

<i>isCommand</i>	If specified in the URL, all occurrences of <code>&</code> and <code>=</code> must be HTTP-encoded as <code>%26</code> and <code>%3D</code> , respectively.
------------------	---

Properties

Optional.

Default

None.

Example

When specified in the viewer URL.

`iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000`

When specified in the config data.

`iscommand=op_sharpen=1&op_colorize=0xff0000`

FavoritesView.maxloadradius

[FavoritesView. | <containerId>_favoritesView.]maxloadradius=-1 | 0 | *preloadnbr*

<i>preloadnbr</i>	<p>Specifies the component preload behavior.</p> <p>When set to <code>-1</code>, all thumbnails are loaded simultaneously when the component is initialized or the asset is changed.</p>
-------------------	--

	<p>When set to 0, only visible thumbnails are loaded.</p> <p>When set to <i>preloadnbr</i>, you can specify how many invisible rows around visible area are preloaded.</p>
--	--

Properties

Optional.

Default

1

Example

maxloadradius=0

FavoritesView.textpos

[FavoritesView. | <containerId>_favoritesView.]textpos=bottom|top|left|right|none|tooltip

bottom top left right none tooltip	<p>Specifies where the label is drawn relative to the thumbnail image. That is, the label is centered at the specified location relative to the thumbnail.</p> <p>When <i>tooltip</i> is specified, the label text is displayed as a floating tooltip over the thumbnail image.</p> <p>When set to <i>none</i>, it turns label display off.</p>
------------------------------------	---

Properties

Optional.

Default

bottom

Example

textpos=top

ImageMapEffect.mapTips

[ImageMapEffect. | <containerId>_imageMapEffect.]mapTips=0|1

0 1	<p>Specifies whether tool tips are enabled for individual map area elements.</p> <p>Ignored on touch devices, including touch-enabled desktop systems.</p>
-----	--

Properties

Optional.

Default

0

Example

mapTips=1

ImageMapEffect.mode

[ImageMapEffect.<containerId>_imageMapEffect.]mode=icon|region|auto|none

icon region auto none	<p>Specifies the appearance of the image map.</p> <ul style="list-style-type: none">• icon map icons are statically shown on the desktop and touch devices.• region renders image map regions; on desktop, they are shown on roll over and on touch devices they are always visible.• auto on desktop systems, image map regions are shown on roll over and on touch devices map icons are always visible.• none disables image maps.
-----------------------	--

Properties

Optional.

Default

icon

Example

mode=auto

ImageMapEffect.rollover

[ImageMapEffect.<containerId>_imageMapEffect.]rollover=0|1

0 1	<p>Specifies when to display the info panel.</p> <p>If set to 1, the info panel is displayed when the mouse enters image map area (in case the image map has non-empty, rollover_key attribute).</p> <p>If set to 0 info panel is triggered when the image map is clicked (if the image map has a non-empty rollover_key and empty href attributes).</p> <p>Ignored on touch devices, including touch-enabled desktop systems, and is automatically set to 0.</p>
-----	---

Properties

Optional.

Default

0


Example

rollover=1

InfoPanelPopup.infoServerUrl

[InfoPanelPopup. | <containerId>_infoPanelPopup.]infoServerUrl=infoserverurl

infoserverurl	<p>Info server URL template is used to fetch key/value pairs for the variable substitution in the info panel content template. The specified template typically contains macro place holders that are replaced with the actual data before the request is sent to the server.</p> <p>\$1\$ is replaced with the rollover value that triggered the InfoPanelPopup activation.</p> <p>\$2\$ is replaced with the sequence number of the current frame in the image set.</p> <p>\$3\$ is replaced with the first path element specified in the name of the parent set of the current item. It typically corresponds to the catalog id.</p> <p>\$4\$ is replaced with the following element in the path and corresponds to the asset id. The actual info server request syntax is info server dependent and it differs from server to server. For example, the following is a typical Scene7 info server request template:</p> <p>http://server_domain/s7info/s7/\$3\$/\$4\$/\$1\$</p>
---------------	--

 **Note:** Be aware that when you configure Info Panel Popup, the HTML code and JavaScript code passed to the Info Panel runs on the client's computer. Therefore, make sure that such HTML code and JavaScript code are secure.

Properties

Optional.

Default

None.

Example

infoServerUrl= http://s7info1.scene7.com/s7info/s7/\$3\$/\$4\$/\$1\$

InfoPanelPopup.showhidetransition

[InfoPanelPopup. | <containerId>_infoPanelPopup.]showhidetranstion=fade|none[, time]

fade none	Specifies the type of info panel show/hide animation.
time	Duration (in seconds) for the show or hide animation.

Properties

Optional.

Default

fade,0.3

Example

showhidetransition=none

InfoPanelPopup.template

[InfoPanelPopup.<containerId>_infoPanelPopup.]template=template

<p>template</p>	<p>The content template that the data returned from the info server is merged into.</p> <p>The content template is an XML following this DTD:</p> <pre><!DOCTYPE info [<!ELEMENT info (var #PCDATA) <!ELEMENT var (#PCDATA)> <!ATTLIST var name CDATA #REQUIRED rollover CDATA #IMPLIED >]></pre> <p>The actual syntax for the content template is the following:</p> <pre><info> <var name='VAR_NAME' rollover='ROLLOVER_KEY'><![CDATA[VAR_VALUE]]> <![CDATA[TEMPLATE_CONTENT]]> </info></pre> <p>That is, the template must start with the <code><info></code> element that may contain optional default <code><var></code> elements. The template content itself, <code>TEMPLATE_CONTENT</code> is HTML text. In addition, the content template may contain variable names enclosed in <code>\$</code> characters that are replaced with the variable values that the info server returns or with default ones.</p> <p>Default variables that are defined in the template can be either global (if rollover attribute is not set) or specific to a certain rollover key (if rollover attribute is present).</p> <p>During template processing variables that are specific to roll over keys take precedence over global variables.</p>
-----------------	--



Note: Be aware that when you configure Info Panel Popup, the HTML code and JavaScript code passed to the Info Panel runs on the client's computer. Therefore, make sure that such HTML code and JavaScript code are secure.

Properties

Optional.

Default

None.

Example

Assuming that the info server response returns the product name as variable \$1\$ and product image URL is returned as variable \$2\$.

```
template=<info><![CDATA[Product description:$1$<br>Product image:]]></info>
```

InitialFrame

```
initialFrame=frame
```

frame	Specifies a zero-based spread index to display on viewer load. The index matches the index of the spread in landscape mode. If the viewer is rotated to portrait, the viewer displays the leftmost page from the spread that is pointed to by frameIdx.
-------	---

Properties

Optional.

Default

0

Example

When specified in the viewer URL.

```
initialFrame=2
```

PageView.doubleclick

```
[PageView.|<containerId>_pageView.]doubleclick=none|zoom|reset|zoomReset
```

none zoom reset zoomReset	Configures the mapping of double-click/tap to zoom actions. Setting to none disables double-click/tap zoom. If set to zoom clicking the image zooms in one zoom step; CTRL+Click zooms out one zoom step. Setting to reset causes a single click on the image to reset the zoom to the initial zoom level. For zoomReset, reset is applied if the current zoom factor is at or beyond the specified limit, otherwise zoom is applied.
---------------------------	---

Properties

Optional.

Default

reset on desktop computers; zoomReset on touch devices.

Example

```
doubleclick=zoom
```

PageView.enableHD

```
[PageView.|<containerId>_pageView.]enableHD=always|never|limit[,number]
```

<code>always never limit</code>	<p>Enable, limit or disable optimization for devices where <code>devicePixelRatio</code> is greater than 1, that is devices with high-density display like iPhone4 and similar devices. If active then the component limits the size of the IS image request as if the device only had a pixel ratio of 1 and that way reducing the bandwidth.</p> <p>See example below</p>
<code>number</code>	<p>If using the limit setting, the component enables high pixel density only up to the specified limit.</p> <p>See example below.</p>

Properties

Optional.

Default

`limit,1500`

Example

The following are the expected results when you use this configuration attribute with the viewer, and the viewer size is 1000 x 1000:

If the set variable equals	Result
<code>always</code>	<p>The pixel density of the screen/device is always taken into account.</p> <ul style="list-style-type: none"> • If the screen pixel density = 1, then the requested image is 1000 x 1000. • If the screen pixel density = 1.5, then the requested image is 1500 x 1500. • If the screen pixel density = 2, then the requested image is 2000 x 2000.
<code>never</code>	<p>It always uses a pixel density of 1 and ignores the device's HD capability. Therefore, the requested image requested is always 1000 x 1000.</p>
<code>limit<number></code>	<p>A device pixel density is requested and served only if the resulting image is below the specified limit.</p> <p>The limit number applies to either the width or the height dimension.</p> <ul style="list-style-type: none"> • If the limit number is 1600, and the pixel density is 1.5, then the 1500 x 1500 image is served.

If the set variable equals	Result
	<ul style="list-style-type: none"> If the limit number is 1600, and the pixel density is 2, then the 1000 x 1000 image is served because the 2000 x 2000 image exceeds the limit. <p>Best practice: The limit number needs to work in conjunction with the company setting for maximum size image. Therefore, set the limit number to equal the company maximum image size setting.</p>

PageView.fmt

[PageView. | <containerId>_pageView.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha

jpg jpeg png png-alpha gif gif-alpha	Specifies the image format that the component uses for loading images from Image Server. If the specified format ends with -alpha, the component renders images as transparent content. For all other image formats the component treats images as opaque. The component has a white background by default. Therefore, to make it transparent, set the background-color CSS property to transparent.
--------------------------------------	--

Properties

Optional.

Default


jpeg

Example

fmt=png-alpha

PageView.frametransition

[PageView. | <containerId>_pageView.]frametransition=slide|turn|auto[,duration]

slide turn auto	<p>Specifies the type of effect that is applied on frame change.</p> <ul style="list-style-type: none"> slide activates a transition where the old frame slides out of view and the new frame slides in to view. turn enables a page flip effect, when a user can drag one of the four spread corners and perform an interactive page flip. <p>When turn is used the appearance of the component is controlled with the pageturnstyle modifier and the .s7pagedivider CSS class is ignored.</p> <p> Note:</p> <p><i>turn animation is not supported on Motorola Xoom.</i></p>
-----------------	---

	<ul style="list-style-type: none"> • <code>auto</code> sets the turn frame transition on desktop systems and the slide transition on touch devices.
<code>duration</code>	Specifies the duration in seconds of a <code>slide</code> or <code>turn</code> transition effect.

Properties

Optional.

Default

`slide,0.3`

Example

```
frametransition=auto,1
```

PageView.iconEffect

```
[PageView. |<containerId>_pageView.] iconeffect=0|1[,count][,fade][,autoHide]
```

<code>0 1</code>	Enables the <code>iconeffect</code> to display on the top of the image when the image is in a reset state and it is suggestive of an available action to interact with the image.
<code>count</code>	Specifies the maximum number of times the <code>iconeffect</code> appears and reappears. A value of <code>-1</code> indicates that the icon always reappears indefinitely.
<code>fade</code>	Specifies the duration of the show or hide animation, in seconds.
<code>autoHide</code>	Sets the number of seconds that the <code>iconeffect</code> stays fully visible before it auto hides. That is, the time after the fade-in animation is completed but before the fade out animation starts. A setting of <code>0</code> disables the auto-hide behavior.

Properties

Optional.

Default


`1,1,0.3,3`

Example

```
iconeffect=0
```

PageView.iscommand

```
[PageView. |<containerId>_pageView.] iscommand=isCommand
```

<i>isCommand</i>	<p>The Image Serving command string that is applied to page image. If specified in the URL all occurrences of & and = must be HTTP-encoded as %26 and %3D, respectively.</p> <p> Note: Image sizing manipulation commands are not supported.</p>
------------------	--

Properties

Optional.

Default

None.

Example

When specified in the viewer URL.

```
iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000
```

When specified in the config data.

```
iscommand=op_sharpen=1&op_colorize=0xff0000
```

PageView.maxloadradius

```
[PageView. | <containerId>_pageView. ]maxloadradius=-1 | 0 | preloadnbr
```

<code>-1 0 preloadnbr</code>	<p>Specifies the component preload behavior.</p> <p>When set to -1 the component preloads all the catalog frames when in an idle state.</p> <p>When set to 0 the component loads only the frame that is currently visible, previous and next frame.</p> <p>Set <i>preloadnbr</i> to define how many invisible frames around the currently displayed frame are preloaded in an idle state.</p>
----------------------------------	---

Properties

Optional.

Default

1

Example

```
maxloadradius=0
```

PageView.pageturnstyle

```
[PageView. | <containerId>_pageView. ]pageturnstyle=dividerWidth,dividerColor,dividerOpacity,borderOnOff,borderColor,fillColor
```

Controls the component appearance when a `PageView.frametransition` is set to `turn` or to `auto` on desktop systems.

<code>dividerWidth</code>	The width in pixels of the page divider shadow that separates the left and right pages in the spread. It also controls the width of the running shadow displayed next to the turning page.
<code>dividerOpacity</code>	The shadow color in RRGGBB format.
<code>dividerOpacity</code>	The shadow opacity in the range of 0 to 1.
<code>borderOnOff</code>	The flag (either 0 or 1) which turns the border around the turning page on and off.
<code>borderColor</code>	The border color in RRGGBB format.
<code>fillColor</code>	The color of the solid fill of the component area used during page turn animation, in RRGGBB format.

Properties

Optional.

Default

40,909090,1,1,909090,FFFFFF

Examples

`pageturnstyle=20,FF0000,1,1,00FF00,0000FF`

PageView.singleclick

`[PageView.<containerId>_pageView.]singleclick=none|zoom|reset|zoomReset`

<code>none zoom reset zoomReset</code>	Configures the mapping of single-click/tap to zoom actions. Setting to <code>none</code> disables single-click/tap zoom. If set to <code>zoom</code> clicking the image zooms in one zoom step; CTRL+Click zooms out one zoom step. Setting to <code>reset</code> causes a single click on the image to reset the zoom to the initial zoom level. For <code>zoomReset</code> , reset is applied if the current zoom factor is at or beyond the specified limit, otherwise zoom is applied.
--	--

Properties

Optional.

Default

`zoomReset` on desktop computers; `none` on touch devices.

Example

`singleclick=zoom`

PageView.transition

```
[PageView. | <containerId>_pageView.]transition=time[, easing]
```

<i>time</i>	Specifies the time in seconds that the animation for a single zoom step action takes.
<i>easing</i>	<p>Creates an illusion of acceleration or deceleration which makes the transition appear more natural. You can set easing to one of the following:</p> <ul style="list-style-type: none">• 0 (auto)• 1 (linear)• 2 (quadratic)• 3 (cubic)• 4 (quartic)• 5 (quintic) <p>Auto mode always uses linear transition when elastic zoom is disabled (default). Otherwise, it fits one of the other easing functions based on the transition time. That is, the shorter the transition time the higher the easing function is used to accelerate the acceleration or deceleration effect.</p>

Properties

Optional.

Default

0.5,0

Example

```
transition=2,2
```

PageView.zoomstep

```
[PageView. | <containerId>_pageView.]zoomstep=step[, limit]
```

<i>step</i>	Configures the number of zoom in and zoom out actions that are required to increase or decrease the resolution by a factor of two. The resolution change for each zoom action is $2^{\pm 1}$ per step. Set to 0 to zoom to full resolution with a single zoom action.
<i>limit</i>	Specifies the maximum zoom resolution, relative to the full resolution image. The default is 1.0, which does not allow zooming beyond full resolution.

Properties

Optional.

Default

1,1

Example

zoomstep=2,3

portraitFrames

portraitFrames=split|solid

split solid	Set to <code>split</code> to let the viewer display double-page spreads as a separate page when used on mobile device in portrait orientation. Set to <code>solid</code> to always display double-page spreads as solid image, irrespective to device orientation.
-------------	--

Properties

Optional.

Default

split

Example

portraitFrames=solid

Print.printquality

[Print.|<containerId>_print.]printquality=*size*

<i>size</i>	The maximum size of the image sent to print.
-------------	--

Properties

Optional.

Default

1000

Example

printquality=800

SocialShare.bearing

[SocialShare.|<containerId>_socialShare.]bearing= up|down|left|right|fit-vertical|fit-lateral

up down left right fit-vertical fit-lateral	Specifies the direction of slide animation for the buttons container.
---	---

	<p>When set to <code>up</code>, <code>down</code>, <code>left</code>, or <code>right</code>, the panel rolls out in a specified direction without an additional bounds check. This behavior can result in panel clipping by an outside container.</p> <p>When set to <code>fit-vertical</code>, the component first shifts the base panel position to the bottom of <code>SocialShare</code> and try to roll out the panel either from the bottom, right, or left, from such base location. With each attempt the component checks to see if panel is clipped by an outside container. If all attempts fail, the component tries to shift the base panel position to the top and repeat the roll out attempts from the top, right, and left direction.</p> <p>When set to <code>fit-lateral</code>, the component uses a similar logic as with <code>fit-vertical</code>, but instead shifts the base to the right first—trying right, down, and up roll out directions—and then shifts the base to the left, trying left, down, and up roll out directions.</p>
--	--

Properties

Optional.

Default

`fit-vertical`

Example

`bearing=left`

TableOfContents.bearing

[TableOfContents.|<containerId>_tableOfContents.]bearing=[fit-lateral|fit-vertical][,autoHideDelay]

<code>fit-lateral fit-vertical</code>	<p>Controls the direction of the drop-down panel appearance.</p> <p>When set to <code>fit-vertical</code>, the component first shifts the base panel position to the bottom of its button and tries to roll out the panel either to the right or to the left from the base location. With each attempt the component checks if the panel is clipped by an outside container. If all attempts fail, the component tries to shift the base panel position to the top and repeat roll out attempts in the right and left direction.</p> <p>When set to <code>fit-lateral</code>, the component uses a similar logic, but shifts the base to the right first, trying down and up roll out directions. Then, it shifts the base to the left, trying down and up roll out directions.</p>
<code>autoHideDelay</code>	<p>Sets the delay in seconds for the drop-down auto-hide timer which hides the panel when a user is idle.</p>

Properties

Optional.

Default

`fit-vertical,2`

Example

```
bearing=fit-vertical,0.5
```

TableOfContents.maxitems

```
[TableOfContents.|<containerId>_tableOfContents.]maxitems=maxitems
```

<i>maxitems</i>	<p>The maximum number of items in the drop-down table of contents.</p> <p>Additionally, you can decrease the number of visible items in the drop-down in case it becomes cropped by the outer container.</p> <p>When set to 0 that component shows as many items as possible given the container it is added to.</p>
-----------------	--

Properties

Optional.

Default

0

Example

```
maxitems=10
```

TableOfContents.showdefault

```
[TableOfContents.|<containerId>_tableOfContents.]showdefault=0|1
```

0 1	<p>When set to 1 the component populates the drop-down panel with elements for all pages, even for those that do not have label defined.</p> <p>When set to 0 only items with explicit labels show in the drop-down panel.</p>
-----	--

Properties

Optional.

Default

1

Example

```
showdefault=0
```

ThumbnailGridView.align

```
[ThumbnailGridView.|<containerId>_gridView.]align=left|center|right
```

left center right	Specifies the internal horizontal alignment (anchoring) of the thumbnails container within the component area. In ThumbnailGridView, the internal thumbnail container is sized so that only a whole number of thumbnails is shown. As a result, there is some padding between the internal container and the external component bounds. This modifier specifies how the internal thumbnails container is positioned horizontally inside the component.
-------------------	--

Properties

Optional.

Default

left

Example

align=center

ThumbnailGridView.enabledragging

[ThumbnailGridView.<containerId>_gridView.]enabledragging=0|1[,overdragvalue]

0 1	Enables or disables the ability for a user to scroll the swatches with a mouse or by using touch gestures
overdragvalue	Functions within the 0–1 range. It is a % value for movement in the wrong direction of the actual speed. If it is set to 1, it moves with the mouse. If it is set to 0, it does not let you move in the wrong direction at all.

Properties

Optional.

Default

1,0.5

Example

enabledragging=0

ThumbnailGridView.fmt

[ThumbnailGridView.<containerId>_gridView.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha

jpg jpeg png png-alpha gif gif-alpha	Specifies the image format that the component uses for loading images from Image Server. It can be any value that Image Server and the client browser supports. If the specified format ends with -alpha, the component renders images as transparent content. For all other image formats the component treats images as opaque.
--------------------------------------	---

Properties

Optional.

Default

jpeg

Example

fmt-png-alpha

ThumbnailGridView.maxloadradius

```
[ThumbnailGridView.<containerId>_pageView.]maxloadradius=-1|0|preloadnbr
```

-1 0 preloadnbr	<p>Specifies the component preload behavior.</p> <p>When set to -1 the thumbnails are loaded simultaneously when the component is initialized or the asset changes.</p> <p>When set to 0 only the visible thumbnails are loaded.</p> <p>Set <i>preloadnbr</i> defines how many invisible rows/columns around the visible area are preloaded.</p>
-----------------	--

Properties

Optional.

Default

1

Example

maxloadradius=0

ThumbnailGridView.scrollbar

```
[ThumbnailGridView.<containerId>_gridView.]scrollbar=0|1
```

0 1	Enables or disable the use of the scroll bar.
-----	---

Properties

Optional.

Default

1

Example

```
scrollbar=0
```

ThumbnailGridView.textpos

```
[ThumbnailGridView.<containerId>_gridView.]textpos=bottom|top|left|right|none|tooltip
```

bottom top left right none tooltip	<p>Specifies where the label is drawn relative to the thumbnail image. That is, the label is centered at the specified location relative to the thumbnail.</p> <p>When <code>tooltip</code> is specified, the label text is displayed as a floating tool tip over the thumbnail image. Set to <code>none</code> to turn off label.</p>
------------------------------------	--

Properties

Optional.

Default

bottom

Example

```
textpos=top
```

Javascript API reference for eCatalog Viewer

The main class of the eCatalog Viewer is `eCatalogViewer`. It is declared in the `s7viewers` namespace. This JavaScript API covers constructor, methods, and callbacks of this particular class.

In all the following examples, `<instance>` stands for the actual name of the JavaScript viewer object that is instantiated from the `s7viewers.eCatalogViewer` class.

dispose

JavaScript API reference for eCatalog Viewer.

```
dispose()
```

Disposes this viewer instance by releasing all resources used by the viewer logic and deleting all inner objects and components created by the viewer in runtime.

The web page code should also delete the viewer instance variable as well to completely remove the viewer from the web browser memory.

If the web page code has registered event listeners directly on Viewer SDK components used by the viewer—or stored external references to such components—such listeners must be explicitly unregistered by the web page code, and such external component references must be deleted prior to calling `dispose()`.

Do not access the Viewer API any more after `dispose()` is called.

Parameters

None.

Returns

None.

Example

```
<instance>.dispose()
```

eCatalogViewer

JavaScript API reference for eCatalog Viewer.

```
eCatalogViewer([config])
```

Constructor, creates a new eCatalog Viewer instance.

Parameters

<i>config</i>	<p>{Object} optional JSON configuration object, allows all the viewer settings to pass to the constructor and avoid calling individual setter methods. Contains the following properties:</p> <ul style="list-style-type: none"> • containerId – {String} ID of the DOM container (normally a <code>DIV</code>) that the viewer is inserted into. It is not necessary to have the container element created by the time this method is called. However, the container must exist when <code>init()</code> is run. Required. • params – {Object} JSON object with viewer configuration parameters where the property name is either a viewer-specific configuration option or an SDK modifier, and the value of that property is a corresponding settings value. Required. • handlers – {Object} JSON object with viewer event callbacks, where the property name is the name of supported viewer event, and the property value is a JavaScript function reference to an appropriate callback. Optional. See Event callbacks for more information about viewer events. • localizedTexts – {Object} JSON object with localization data. Optional. See Localization of user interface elements for more information. <p>See also the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
---------------	---

Returns

None.

Example

```
var eCatalogViewer = new s7viewers.eCatalogViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Viewers/Pluralist",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  },
  "handlers": {
    "initComplete": function() {
      console.log("init complete");
    }
  }
},
```

```

"localizedTexts":{
"en":{
"CloseButton.TOOLTIP":"Close"
},
"fr":{
"CloseButton.TOOLTIP":"Fermer"
},
defaultLocale:"en"
}
});

```

getComponent

JavaScript API reference for eCatalog Viewer

`getComponent(componentId)`

Returns a reference to the Viewer SDK component that is used by the viewer. The web page can use this method to extend or customize the behavior of the out-of-box viewer. Call this method only after the `initComplete` viewer callback has run, otherwise the component may not be created yet by the viewer logic.

Parameters

componentID - {String} an ID of the Viewer SDK component used by the viewer. This viewer supports the following component IDs:

Component ID	Viewer SDK component class name
<code>parameterManager</code>	<code>s7sdk.ParameterManager</code>
<code>container</code>	<code>s7sdk.common.Container</code>
<code>mediaSet</code>	<code>s7sdk.set.MediaSet</code>
<code>pageView</code>	<code>s7sdk.set.PageView</code>
<code>primaryControls</code>	<code>s7sdk.common.ControlBar</code>
<code>secondaryControls</code>	<code>s7sdk.common.ControlBar</code>
<code>gridView</code>	<code>s7sdk.set.ThumbnailGridView</code>
<code>tableOfContents</code>	<code>s7sdk.set.TableOfContents</code>
<code>infoPanelPopup</code>	<code>s7sdk.info.InfoPanelPopup</code>
<code>imageMapEffect</code>	<code>s7sdk.image.ImageMapEffect</code>
<code>leftButton</code>	<code>s7sdk.common.PanLeftButton</code>
<code>rightButton</code>	<code>s7sdk.common.PanRightButton</code>

Component ID	Viewer SDK component class name
zoomInButton	s7sdk.common.ZoomInButton
zoomOutButton	s7sdk.common.ZoomOutButton
zoomResetButton	s7sdk.common.ZoomResetButton
secondaryZoomResetButton	s7sdk.common.ZoomResetButton
thumbnailPageButton	s7sdk.common.ThumbnailPageButton
fullScreenButton	s7sdk.common.FullScreenButton
toolBarLeftButton	s7sdk.common.PanLeftButton
toolBarRightButton	s7sdk.common.PanRightButton
firstPageButton	s7sdk.common.PanLeftButton
secondaryFirstPageButton	s7sdk.common.PanLeftButton
lastPageButton	s7sdk.common.PanRightButton
secondaryLastPageButton	s7sdk.common.PanRightButton
closeButton	s7sdk.common.CloseButton
socialShare	s7sdk.share.SocialShare
twitterShare	s7sdk.share.TwitterShare
facebookShare	s7sdk.share.FacebookShare
linkShare	s7sdk.share.LinkShare
emailShare	s7sdk.share.EmailShare
embedShare	s7sdk.share.EmbedShare
print	s7sdk.share.Print
download	s7sdk.common.Download

Component ID	Viewer SDK component class name
favoritesEffect	s7sdk.favorites.FavoritesEffect
favoritesView	s7sdk.favorites.FavoritesView
favoritesMenu	s7sdk.favorites.FavoritesMenu
addFavoriteButton	s7sdk.favorites.AddFavoriteButton
removeFavoriteButton	s7sdk.favorites.RemoveFavoriteButton
viewAllFavoriteButton	s7sdk.favorites.ViewAllFavoriteButton

When working with SDK APIs it is important to use correct fully qualified SDK namespace as described in [Viewer SDK namespace](#). See the *Viewer SDK API* documentation for more information about a particular component.

Returns

{Object} a reference to Viewer SDK component. The method returns null if the `componentId` is not a supported viewer component or if the component was not yet created by the viewer logic.

Example

```
<instance>.setHandlers({  
  "initComplete":function() {  
    var pageView = <instance>.getComponent("pageView");  
  }  
})
```

init

JavaScript API reference for eCatalog Viewer.

`init()`

Starts the initialization of the eCatalog Viewer. By this time container DOM element must be created so that the viewer code can find it by its ID.

If the container element is not a part of the web page layout just yet (for example, it may be hidden using `display:none` style assigned to it), the viewer suspends its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load automatically resumes.

Only call this method once during the viewer life cycle; subsequent calls are ignored.

Parameters

None.

Returns

{Object} A reference to viewer instance.

Example

```
\<instance>.init()
```

setAsset

JavaScript API reference for Video Viewer.

```
setAsset(asset)
```

asset	<p>{String} new asset id or explicit image set with optional Image Serving modifiers appended after ?.</p> <p>Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.</p>
-------	---

Sets a new asset. You can call this parameter at any time, either before or after `init()`. If it is called after `init()`, the viewer swaps the asset at runtime.

See also [init](#).

Returns

None.

Example

Single reference to an image set that is defined in a catalog:

```
<instance>.setAsset("Viewers/Pluralist")
```

Explicit image set, with pre-combined pages:

```
<instance>.setAsset("Scene7SharedAssets/Backpack_B,Scene7SharedAssets/Backpack_C,Scene7SharedAssets/Backpack_H,Scene7SharedAssets/Backpack_U")
```

Explicit image set, with individual page images:

```
<instance>.setAsset("Scene7SharedAssets/AdobeScene7OverviewUS1,Scene7SharedAssets/AdobeScene7OverviewUS2,AdobeScene7OverviewUS3,Scene7SharedAssets/AdobeScene7OverviewUS4")
```

Sharpening modifier added to all images in the set:

```
<instance>.setAsset("Viewers/Pluralist?op_sharpen=1")
```

setContainerId

JavaScript API reference for eCatalog Viewer.

```
setContainerId(containerId)
```

Sets the ID of the DOM container (normally a DIV) into which the viewer is inserted. It is not necessary to have the container element created by the time this method is called. However, the container must exist when `init()` is run. It must be called before `init()`.

This method is optional if the viewer configuration information is passed with `config` JSON object to the constructor.

containerId	{string} ID of container.
-------------	---------------------------

Returns

None.

Example

```
<instance>.setContainerId("s7viewer");
```

setHandlers

JavaScript API reference for eCatalog Viewer.

```
setHandlers(handlers)
```

Specifies zero or more callback handlers. A call to this method fully overwrites event handlers that were previously assigned for that viewer instance. Must be called before `init()`.

Parameter

<i>handlers</i>	<p>{Object} JSON object with viewer event callbacks, where the property name is the name of the supported viewer event and the property value is a JavaScript function reference to an appropriate callback.</p> <p>See Event callbacks for more information about viewer events.</p>
-----------------	---

Returns

None.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    console.log("init complete");
  }
})
```

setLocalizedTexts

JavaScript API reference for Video Viewer.

```
setLocalizedTexts(localizationInfo)
```

<i>localizationInfo</i>	<p>{Object} JSON object with localization data.</p> <p>See Localization of user interface elements for more information.</p> <p>See also the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
-------------------------	--

Sets localization SYMBOL values for one or more locales. This parameter must be called before `init()`.

See also [init](#).

Returns

None.

Example

```
<instance>.setLocalizedTexts({ "en": { "CloseButton.TOOLTIP": "Close" }, "fr": { "CloseButton.TOOLTIP": "Fermer" }, defaultLocale: "en" })
```

setParam

JavaScript API reference for eCatalog Viewer.

```
setParam(name, value)
```

Sets the viewer parameter to a specified value. The parameter is either a viewer-specific configuration option or a software development kit modifier. This parameter is called before `init()`.

This method is optional if the viewer configuration information is passed with `config` JSON object to the constructor.

See also [init](#).

<i>name</i>	{string} name of parameter.
<i>value</i>	{string} value of parameter. The value cannot be percent-encoded.

Returns

None.

Example

```
<instance>.setParam("style", "customStyle.css")
```

setParams

JavaScript API reference for eCatalog Viewer.

```
setParams(params)
```

Sets one or more parameters to a given value. The method argument syntax is identical to a URL query string. That is, it represents name=value pairs separated with `&`. Just as in a query string, the names and values are percent-encoded using UTF8. Before you call `init()`, this parameter must be called.

This method is optional if the viewer configuration information is passed with `config` JSON object to the constructor.

See also [init](#).

<i>params</i>	{string} name=value parameter pairs separated with <code>&</code> .
---------------	---

Returns

None.

Example

```
<instance>.setParams("PageView.zoomstep=2,3&PageView.iscommand=op_sharpen%3d1")
```

Event callbacks

The viewer supports JavaScript event callbacks that the web page uses to track the viewer initialization process or runtime behavior.

Callback handlers are assigned by passing event names and corresponding handler functions with the `handlers` property to `config` JSON object in the viewer's constructor. Alternatively, it is possible to use `setHandlers()` API method.

Supported viewer events include the following:

- `initComplete` – triggers when viewer initialization is complete and all internal components are created, so that it is possible to use `getComponent()` API. The callback handler does not take any arguments.
- `trackEvent` – triggers each time an event occurs inside the viewer which may be handled by an event tracking system, such as Adobe Analytics. The callback handler takes the following arguments:
 - `objID` {String} not currently used.
 - `compClass` {String} not currently used.
 - `instName` {String} an instance name of the Viewer SDK component that triggered the event.
 - `timeStamp` {Number} event time stamp.
 - `eventInfo` {String} event payload.

See also [eCatalogViewer](#) and [setHandlers](#).

Customizing eCatalog Viewer

All visual customization and most behavior customization for the eCatalog Viewer is done by creating a custom CSS.

The suggested workflow is to take the default CSS file for the appropriate viewer, copy it to a different location, customize it, and specify the location of the customized file in the `style=` command.

Default CSS files can be found at the following location:

```
<s7_viewers_root>/html5/eCatalogViewer_dark.css
```

The custom CSS file must contain the same class declarations as the default one. If a class declaration is omitted, the viewer does not function properly because it does not provide built-in default styles for the user interface elements.

An alternative way to provide custom CSS rules is to use embedded styles directly on the web page or in one of the linked external CSS rules.

When creating custom CSS keep in mind that the viewer assigns `.s7ecatalogviewer` class to its container DOM element. If you are using external CSS file passed with the `style=` command, use `.s7ecatalogviewer` class as parent class in descendant selector for your CSS rules. If you are doing embedded styles on the web page, additionally qualify this selector with an ID of the container DOM element as follows:

```
#<containerId>.s7ecatalogviewer
```

Building responsive designed CSS

It is possible to target different devices and embedding sizes in CSS to make your content display differently, depending on a user's device or a particular web page layout. This includes, but is not limited to, different web page layouts, user interface element sizes, and artwork resolution.

The viewer supports two methods for creating responsive designed CSS: CSS markers and standard CSS media queries. You can use these methods independently or together.

CSS markers

To assist in creating responsive designed CSS, the viewer supports CSS markers which special CSS classes dynamically assigned to the top-level viewer container element based on the run-time viewer size and the input type used on the current device.

The first group of CSS markers includes `.s7size_large`, `.s7size_medium`, and `.s7size_small` classes. They are applied based on the runtime area of the viewer container. That is, if the viewer area is equal to or bigger than the size of a common desktop monitor `.s7size_large` is used; if the area is close in size to a common tablet device `.s7size_medium` is assigned. For areas similar to mobile phone screens `.s7size_small` is set. The primary purpose of these CSS markers is to create different user interface layouts for different screens and viewer sizes.

The second group of CSS Markers includes `.s7mouseinput` and `.s7touchinput`. `.s7touchinput` is set if the current device has touch input capabilities; otherwise, `.s7mouseinput` is used. These markers are intended to create user interface input elements with different screen sizes for different input types, because normally touch input requires larger elements. In case the device has both mouse input and touch capabilities, `.s7touchinput` is set and the viewer renders a touch-friendly user interface.

The following sample CSS sets the zoom in button size to 28 x 28 pixels on systems with mouse input, and 56 x 56 pixels on touch devices. In addition, it hides the button completely if the viewer size becomes really small:

```
.s7ecatalogviewer.s7mouseinput .s7zoominbutton {
    width:28px;
    height:28px;
}
.s7ecatalogviewer.s7touchinput .s7zoominbutton {
    width:56px;
    height:56px;
}
.s7ecatalogviewer.s7size_small .s7zoominbutton {
    visibility:hidden;
}
```

To target devices with a different pixel density, use CSS media queries. The following media query block would contain CSS that is specific to high density screens:

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5)
{
}
```

Using CSS markers is the most flexible way of building responsive designed CSS as it allows you to target not only device screen size but actual viewer size, which may be useful for responsive designed page layouts.

Use the default viewer CSS file as an example of a CSS markers approach.

CSS media queries

Device sensing can also be done using pure CSS media queries. Everything enclosed within a given media query block is applied only when it is run on a corresponding device.

When applied to Mobile Viewers, use four CSS media queries, defined in your CSS in the following order:

1. Contains only rules specific for all touch devices.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px)
{
}
```

2. Contains only rules specific for tablets with high resolution screens.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px) and (-webkit-min-device-pixel-ratio:1.5),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px) and (-webkit-min-device-pixel-ratio:1.5)
{
}
```

3. Contains only rules specific for all mobile phones.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in)
{
}
```

4. Contains only rules specific for mobile phones with high resolution screens.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in) and
(-webkit-min-device-pixel-ratio: 1.5),
only screen and (device-width:720px) and (device-height:1280px) and
(-webkit-device-pixel-ratio: 2),
only screen and (device-width:1280px) and (device-height:720px) and
(-webkit-device-pixel-ratio: 2)
{
}
```

Using a media queries approach, you should organize CSS with device sensing as follows:

- First, the desktop-specific section defines all properties that are either desktop-specific or common to all screens.
- And second, the four media queries go in the order defined above and provide CSS rules that are specific for the corresponding device type.

There is no need to duplicate the entire viewer CSS in each media query. Only properties that are specific to given devices are redefined inside a media query.

CSS Sprites

Many viewer user interface elements are styled using bitmap artwork and have more than one distinct visual state. A good example is a button that normally has at least three different states: "up", "over", and "down". Each state requires its own bitmap artwork assigned.

With a classic approach to styling, the CSS would have a separate reference to individual image file on the server for each state of the user interface element. The following is a sample CSS for styling a zoom-in button:

```
.s7ecatalogviewer.s7mouseinput .s7zoominbutton[state='up'] {
background-image:url(images/v2/ZoomInButton_dark_up.png);
}
.s7ecatalogviewer.s7mouseinput .s7zoominbutton[state='over'] {
background-image:url(images/v2/ZoomInButton_dark_over.png);
}
.s7ecatalogviewer.s7mouseinput .s7zoominbutton[state='down'] {
background-image:url(images/v2/ZoomInButton_dark_down.png);
}
.s7ecatalogviewer.s7mouseinput .s7zoominbutton[state='disabled'] {
background-image:url(images/v2/ZoomInButton_dark_disabled.png);
}
```

The drawback to this approach is that the end user experiences flickering or delayed user interface response when the element is interacted with for the first time. This action occurs because the image artwork for the new element state is not yet downloaded. Also, this approach may have a slight negative impact on performance because of an increase in the number of HTTP calls to the server.

CSS sprites is a different approach where image artwork for all element states is combined into a single PNG file called a "sprite". Such "sprite" has all visual states for the given element positioned one after another. When styling a user interface element with sprites the same sprite image is referenced for all different states in the CSS. Also, the `background-position` property is used for each state to specify which part of the "sprite" image is used. You can structure a "sprite" image in any suitable way. Viewers normally have it vertically stacked. Below is a "sprite"-based example of styling the same zoom-in button from above:

```
.s7ecatalogviewer .s7zoominbutton[state] {
    background-image: url(images/v2/ZoomInButton_dark_sprite.png);
}
.s7ecatalogviewer.s7mouseinput .s7zoominbutton[state='up'] {
background-position: -84px -560px;
}
.s7ecatalogviewer.s7mouseinput .s7zoominbutton[state='over'] {
background-position: -56px -560px;
}
.s7ecatalogviewer.s7mouseinput .s7zoominbutton[state='down'] {
background-position: -28px -560px;
}
.s7ecatalogviewer.s7mouseinput .s7zoominbutton[state='disabled'] {
background-position: -0px -560px;
}
```

General styling notes and advice

- When customizing the viewer user interface with CSS the use of the `!important` rule is not supported to style viewer elements. In particular, `!important` rule should not be used to override any default or run-time styling provided by the viewer or Viewer SDK. The reason is that it may affect the behavior of proper components. Instead, you should use CSS selectors with the proper specificity to set CSS properties that are documented in this reference guide.
- All paths to external assets within CSS are resolved against the CSS location, not the viewer HTML page location. Be aware of this rule when you copy the default CSS to a different location. Either copy the default assets as well or update paths within the custom CSS.
- The preferred format for bitmap artwork is PNG.
- Bitmap artwork is assigned to user interface elements using the `background-image` property.
- The `width` and `height` properties of a user interface element define its logical size. The size of the bitmap passed to `background-image` does not affect logical size.
- To use the high pixel density of high-resolution screens like Retina, specify bitmap artwork twice as large as the logical user interface element size. Then, apply the `-webkit-background-size: contain` property to scale the background down to the logical user interface element size.
- To remove a button from the user interface, add `display: none` to its CSS class.
- You can use various formats for color value that CSS supports. If you need transparency, use the format `rgba(R,G,B,A)`. Otherwise, you can use the format `#RRGGBB`.

Common User Interface Elements

The following is user interface elements reference documentation that applies to eCatalog Viewer:

Add Favorite button

The position of the Add Favorite button is fully managed by the Favorites menu.

The appearance of the Add Favorite button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7addfavoritebutton
```


CSS properties of the Add Favorite button

background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
width	Width of the button.
height	Height of the button.



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the state when a user can add a new Favorite icon by clicking or tapping. *selected='false'* corresponds to the normal operation mode when a user can zoom, pan, and swap pages.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Add Favorite button that is 28 x 28 pixels, and displays a different image for each of the four different button states when selected or not selected.

```
.s7ecatalogviewer .s7addfavoritebutton {
  width:28px;
  height:28px;
}
.s7ecatalogviewer .s7addfavoritebutton[selected='false'][state='up'] {
  background-image:url(images/v2/AddFavoriteButton_dark_up.png);
}
.s7ecatalogviewer .s7addfavoritebutton[selected='false'][state='over'] {
  background-image:url(images/v2/AddFavoriteButton_dark_over.png);
}
.s7ecatalogviewer .s7addfavoritebutton[selected='false'][state='down'] {
  background-image:url(images/v2/AddFavoriteButton_dark_down.png);
}
.s7ecatalogviewer .s7addfavoritebutton[selected='false'][state='disabled'] {
  background-image:url(images/v2/AddFavoriteButton_dark_disabled.png);
}
.s7ecatalogviewer .s7addfavoritebutton[selected='true'][state='up'] {
  background-image:url(images/v2/AddFavoriteButton_dark_over.png);
}
.s7ecatalogviewer .s7addfavoritebutton[selected='true'][state='over'] {
  background-image:url(images/v2/AddFavoriteButton_dark_over.png);
}
.s7ecatalogviewer .s7addfavoritebutton[selected='true'][state='down'] {
  background-image:url(images/v2/AddFavoriteButton_dark_over.png);
}
.s7ecatalogviewer .s7addfavoritebutton[selected='true'][state='disabled'] {
  background-image:url(images/v2/AddFavoriteButton_dark_disabled.png);
}
```

Close button

Clicking or tapping this button closes the containing web page. This button only appears if the `closebutton` parameter is set to 1. This button is not available on desktop systems. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7closebutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.
left	Position from the left border of the main control bar, including padding.
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which you can use to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a close button that is 56 x 56 pixels, positioned 4 pixels from the top and right edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7closebutton {
top:4px;
right:4px;
width:56px;
height:56px;
}
.s7ecatalogviewer .s7closebutton [state='up'] {
background-image:url(images/v2/CloseButton_dark_up.png);
}
.s7ecatalogviewer .s7closebutton [state='over'] {
background-image:url(images/v2/CloseButton_dark_over.png);
}
.s7ecatalogviewer .s7closebutton [state='down'] {
background-image:url(images/v2/CloseButton_dark_down.png);
}
.s7ecatalogviewer .s7closebutton [state='disabled'] {
background-image:url(images/v2/CloseButton_dark_disabled.png);
}
```

Download

The appearance of the Download button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7download
```

CSS properties of the Download button

margin-top	The offset from the top of the control bar.
margin-left	The distance to the next button on the left, or the left side of the control bar if this is the first button in a row.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Download button that is 28 x 28 pixels and displays a different image for each of the four different button states:

```
.s7ecatalogviewer .s7download {
  margin-top: 4px;
  margin-left: 10px;
  width: 28px;
  height: 28px;
}
.s7ecatalogviewer .s7download[state='up'] {
  background-image: url(images/v2/Download_dark_up.png);
}
.s7ecatalogviewer .s7download[state='over'] {
  background-image: url(images/v2/Download_dark_over.png);
}
.s7ecatalogviewer .s7download[state='down'] {
  background-image: url(images/v2/Download_dark_down.png);
}
.s7ecatalogviewer .s7download[state='disabled'] {
  background-image: url(images/v2/Download_dark_disabled.png);
}
```

Email share

Email share tool consists of a button added to the Social share panel and the modal dialog box which displays when the tool is activated. The position of the button is fully managed by the Social share tool.

The appearance of the email share button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emailshare
```

CSS properties of the email share tool

width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a email share button that is 28 x 28 pixels, and that displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7emailshare {
  width:28px;
  height:28px;
}
.s7ecatalogviewer .s7emailshare[state='up'] {
  background-image:url(images/v2/EmailShare_dark_up.png);
}
.s7ecatalogviewer .s7emailshare[state='over'] {
  background-image:url(images/v2/EmailShare_dark_over.png);
}
.s7ecatalogviewer .s7emailshare[state='down'] {
  background-image:url(images/v2/EmailShare_dark_down.png);
}
.s7ecatalogviewer .s7emailshare[state='disabled'] {
  background-image:url(images/v2/EmailShare_dark_disabled.png);
}
```

The background overlay which covers web page when the dialog is active is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7backoverlay
```

CSS properties of the back overlay

opacity	Background overlay opacity.
background-color	Background overlay color.

Example – to set up background overlay to be gray with 70% opacity:

```
.s7ecatalogviewer .s7emaildialog .s7backoverlay {
  opacity:0.7;
  background-color:#222222;
}
```

By default the modal dialog is displayed centered in the screen on desktop systems and takes the whole web page area on touch devices. In all cases, the positioning and sizing of the dialog box is managed by the component. The dialog is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialog
```

CSS properties of the dialog box

border-radius	Dialog box border radius (in case the dialog box does not take the entire browser window);
background-color	Dialog box background color;
width	Should be either unset, or set to 100%, in which case the dialog takes the whole browser window (this mode is preferred on touch devices);
height	Should be either unset, or set to 100%, in which case the dialog takes the whole browser window (this mode is preferred on touch devices).

Example – to set up dialog to use the entire browser window and have a white background on touch devices:

```
.s7ecatalogviewer .s7touchinput .s7emaildialog .s7dialog {
  width:100%;
  height:100%;
  background-color: #ffffff;
}
```

The dialog box header consists of an icon, a title text and a close button. The header container is controlled with the following CSS class selector

```
.s7ecatalogviewer .s7emaildialog .s7dialogheader
```

CSS properties of the dialog box header

padding	Inner padding for header content.
---------	-----------------------------------

The icon and the title text are wrapped into an additional container controlled with

```
.s7ecatalogviewer .s7emaildialog .s7dialogheader .s7dialogline
```

CSS properties of the dialog line

padding	Inner padding for the header icon and title.
---------	--

Header icon is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogheadericon
```

CSS properties of the dialog box header icon

width	Icon width.
height	Icon height.

background-image	Icon image.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .

Header title is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogheadertext
```

CSS properties of the dialog box header text

font-weight	Font weight.
font-size	Font height.
font-family	Font family.
padding	Internal text padding.

Close button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7closebutton
```

CSS properties of the close button

top	Vertical button position relative to header container.
right	Horizontal button position relative to header container.
width	Button width.
height	Button height.
padding	Inner padding of button.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The Close button tool tip and the dialog box title can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up dialog header with padding, 24 x 17 pixels icon, bold 16 point title, and a 28 x 28 pixel Close button positioned two pixels from the top and two pixels from the right of dialog box container:

```
.s7ecatalogviewer .s7emaildialog .s7dialogheader {
padding: 10px;
}
.s7ecatalogviewer .s7emaildialog .s7dialogheader .s7dialogline {
padding: 10px 10px 2px;
}
.s7ecatalogviewer .s7emaildialog .s7dialogheadericon {
background-image: url("images/sdk/dlgemail_cap.png");
height: 17px;
width: 24px;
}
.s7ecatalogviewer .s7emaildialog .s7dialogheadertext {
font-size: 16pt;
font-weight: bold;
padding-left: 16px;
}
.s7ecatalogviewer .s7emaildialog .s7closebutton {
top: 2px;
right: 2px;
padding: 8px;
width: 28px;
height: 28px;
}
.s7ecatalogviewer .s7emaildialog .s7closebutton[state='up'] {
background-image: url(images/sdk/close_up.png);
}
.s7ecatalogviewer .s7emaildialog .s7closebutton[state='over'] {
background-image: url(images/sdk/close_over.png);
}
.s7ecatalogviewer .s7emaildialog .s7closebutton[state='down'] {
background-image: url(images/sdk/close_down.png);
}
.s7ecatalogviewer .s7emaildialog .s7closebutton[state='disabled'] {
background-image: url(images/sdk/close_disabled.png);
}
```

Dialog footer consists of Cancel and Send email buttons. The footer container is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogfooter
```

CSS properties of the dialog box footer

border	Border that you can use to visually separate the footer from the rest of the dialog box.
--------	--

The footer has an inner container which keeps both buttons. It is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogbuttoncontainer
```

CSS properties of the dialog box button container

padding	Inner padding between the footer and the buttons.
---------	---

Cancel button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogcancelbutton
```

CSS properties of the dialog box cancel button

width	Button width.
-------	---------------

height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

Send email button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogactionbutton
```

CSS properties of the dialog box action button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

In addition, both buttons share the same common CSS class which can contain CSS settings that are the same for other dialog box buttons:

```
.s7ecatalogviewer .s7emaildialog .s7dialogfooter .s7button
```

CSS properties of the button

font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.
line-height	Text height inside the button. Affects vertical alignment.
box-shadow	Drop shadow.
margin-right	Right button margin.

This buttons tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box footer with 64 x 34 Cancel button and a 82 x 34 send email button, with the text color and background color different for each button state:

```
.s7ecatalogviewer .s7emaildialog .s7dialogfooter {
  border-top: 1px solid #909090;
}
.s7ecatalogviewer .s7emaildialog .s7dialogbuttoncontainer {
  padding-bottom: 6px;
  padding-top: 10px;
}
.s7ecatalogviewer .s7emaildialog .s7dialogfooter .s7button {
  box-shadow: 1px 1px 1px #999999;
  color: #FFFFFF;
  font-size: 9pt;
  font-weight: bold;
  line-height: 34px;
  margin-right: 10px;
}
.s7ecatalogviewer .s7emaildialog .s7dialogcancelbutton {
  width: 64px;
  height: 34px;
}
.s7ecatalogviewer .s7emaildialog .s7dialogcancelbutton[state='up'] {
  background-color: #666666;
  color: #dddddd;
}
.s7ecatalogviewer .s7emaildialog .s7dialogcancelbutton[state='down'] {
  background-color: #555555;
  color: #ffffff;
}
.s7ecatalogviewer .s7emaildialog .s7dialogcancelbutton[state='over'] {
  background-color: #555555;
  color: #ffffff;
}
.s7ecatalogviewer .s7emaildialog .s7dialogcancelbutton[state='disabled'] {
  background-color: #b2b2b2;
  color: #dddddd;
}
.s7ecatalogviewer .s7emaildialog .s7dialogactionbutton {
  width: 82px;
  height: 34px;
}
.s7ecatalogviewer .s7emaildialog .s7dialogactionbutton[state='up'] {
  background-color: #333333;
  color: #dddddd;
}
.s7ecatalogviewer .s7emaildialog .s7dialogactionbutton[state='down'] {
  background-color: #222222;
  color: #cccccc;
}
.s7ecatalogviewer .s7emaildialog .s7dialogactionbutton[state='over'] {
  background-color: #222222;
  color: #cccccc;
}
.s7ecatalogviewer .s7emaildialog .s7dialogactionbutton[state='disabled'] {
  background-color: #b2b2b2;
  color: #dddddd;
}
```

The main dialog area (between the header and the footer) contains scrollable dialog content and scroll panel on the right. In all cases, the component manages the width of this area, it is not possible to set it in CSS. Main dialog area is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogviewarea
```

CSS properties of the dialog box viewing area

height	The height of the main dialog box area. It should be specified only when the dialog box works in desktop mode. It is not applicable when the dialog box is sized to occupy the entire browser window.
background-color	The background color of the main dialog box area.
margin	Outer margin.



Note: The main dialog box area supports the optional *state* attribute selector. It is set to *sendsuccess* when the email form is submitted and the dialog box shows a confirmation message. As long as the confirmation message is small, this attribute selector can be used to reduce the dialog box height when such confirmation message is displayed.

Example – to set up the main dialog box area to be 300 pixels height initially and 100 pixels height when the confirmation message is shown, have a ten pixel margin, and use a white background:

```
.s7ecatalogviewer .s7emaildialog .s7dialogviewarea {
  background-color:#ffffff;
  margin:10px;
  height:300px;
}
.s7ecatalogviewer .s7emaildialog .s7dialogviewarea[state='sendsuccess'] {
  height:100px;
}
```

All form content (like labels and input fields) resides inside a container controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogbody
```

If the height of this container appears to be bigger than the main dialog box area, a vertical scroll is enabled automatically by the component.

CSS properties of the dialog box body

padding	Inner padding.
---------	----------------

Example – to set up form content to have ten pixel padding:

```
.s7ecatalogviewer .s7emaildialog .s7dialogbody {
  padding: 10px;
}
```

Dialog box form is filled on line-by-line basis, where each line carries a part of the form content (like a label and a text input field). Single form line is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogbody .s7dialogline
```

CSS properties of the dialog box line

padding	Inner line padding.
---------	---------------------

Example – to set up a dialog box form to have ten pixel padding for each line:

```
.s7ecatalogviewer .s7emaildialog .s7dialogbody .s7dialogline {
  padding: 10px;
}
```

All static labels in the dialog box form are controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialoglabel
```

This class is not suitable for controlling labels size or position because you can apply it to texts in various places of the form user interface.

CSS properties of the dialog box label.

font-weight	Label font weight.
font-size	Label font size.
font-family	Label font family.
color	Label text color.

Dialog box labels can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up all labels to be gray, bold, with a nine pixel font:

```
.s7ecatalogviewer .s7emaildialog .s7dialoglabel {
  color: #666666;
  font-size: 9pt;
  font-weight: bold;
}
```

All static labels that are displayed to the left of the form input fields are controlled with:

```
.s7ecatalogviewer .s7emaildialog .s7dialoginputlabel
```

CSS properties of the dialog box input label

width	The width of the static label.
text-align	The horizontal text alignment.
margin	Static label margin.
padding	Static label padding.

Example – to set up input field labels to be 50 pixels width, right-aligned, have ten pixels of padding, and a ten pixel margin on the right:

```
.s7ecatalogviewer .s7emaildialog .s7dialoginputlabel {
  margin-right: 10px;
  padding: 10px;
  text-align: right;
  width: 50px;
}
```

Each form input field is wrapped into the container which lets you apply a custom border around the input field. It is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

border	Border around the input field container.
padding	Inner padding.



Note: Input field container supports optional *state* attribute selector. It is set to *verifyerror* when the user makes a mistake in input data format and inline validation fails. This attribute selector can be used to highlight incorrect user input in the form.

Most input fields that spread from the label on the left up to the right edge of the dialog box body (which includes From field and Message field) are controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialoginputwide
```

CSS properties of the dialog box input wide field

width	Input field width.
-------	--------------------

The To field is narrower because it allocates space for the Remove email button on the right. It is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialoginputshort
```

CSS properties of the dialog box input short field

width	Input field width.
-------	--------------------

Example – to set up a form to have a one pixel grey border with nine pixels of padding around all input fields; to have the same border in red color for fields which fail validation, to have 250 pixels wide To field, and the rest of the input fields 300 pixels wide:

```
.s7ecatalogviewer .s7emaildialog .s7dialoginputcontainer {
  border: 1px solid #CCCCCC;
  padding: 9px;
}
.s7ecatalogviewer .s7emaildialog .s7dialoginputcontainer[state="verifyerror"] {
  border: 1px solid #FF0000;
}
.s7ecatalogviewer .s7emaildialog .s7dialoginputshort {
  width: 250px;
}
.s7ecatalogviewer .s7emaildialog .s7dialoginputwide {
  width: 300px;
}
```

Email message input field is additionally controlled with:

```
.s7ecatalogviewer .s7emaildialog .s7dialogmessage
```

This class lets you set specific properties for the underlying TEXTAREA element.

CSS properties of the dialog box message

height	Message height.
word-wrap	Word wrapping style.

Example – to set up an email message to be 50 pixels high and use break-word word wrapping:

```
.s7ecatalogviewer .s7emaildialog .s7dialogmessage {
    height: 50px;
    word-wrap: break-word;
}
```

Add Another Email Address button lets a user add more than one addressee in email form. It is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogaddemailbutton
```

CSS properties of the dialog box add email address button

height	Button height.
color	Button text color for each state.
background-image	Button image for each state.
background-position	Button image position inside the button area.
font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.
line-height	Text height inside the button. Affects the vertical alignment.
text-align	Horizontal text alignment.
padding	Inner padding.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up "Add Another Email Address" button to be 25 pixels high, use 12 point bold font with right alignment, and a different text color and image for each state:

```
.s7ecatalogviewer .s7emaildialog .s7dialogaddemailbutton {
    text-align:right;
    font-size:12pt;
    font-weight:bold;
    background-position:left center;
    line-height:25px;
    padding-left:30px;
    height:25px;
}
.s7ecatalogviewer .s7emaildialog .s7dialogaddemailbutton[state='up'] {
    color:#666666;
    background-image:url(images/sdk/dlgaddplus_up.png);
}
.s7ecatalogviewer .s7emaildialog .s7dialogaddemailbutton[state='down'] {
```

```

color:#000000;
background-image:url(images/sdk/dlgaddplus_over.png);
}
.s7ecatalogviewer .s7emaildialog .s7dialogaddemailbutton[state='over'] {
color:#000000;
text-decoration:underline;
background-image:url(images/sdk/dlgaddplus_over.png);
}
.s7ecatalogviewer .s7emaildialog .s7dialogaddemailbutton[state='disabled'] {
color:#666666;
background-image:url(images/sdk/dlgaddplus_up.png);
}

```

Remove button lets a user remove extra addressees from the email form. It is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogremoveemailbutton
```

CSS properties of the dialog box remove email button

width	Button width.
height	Button height.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Remove button to be 25 x 25 pixels and use a different image for each state:

```

.s7ecatalogviewer .s7emaildialog .s7dialogremoveemailbutton {
width:25px;
height:25px;
}
.s7ecatalogviewer .s7emaildialog .s7dialogremoveemailbutton[state='up'] {
background-image:url(images/sdk/dlgremove_up.png);
}
.s7ecatalogviewer .s7emaildialog .s7dialogremoveemailbutton[state='over'] {
background-image:url(images/sdk/dlgremove_over.png);
}
.s7ecatalogviewer .s7emaildialog .s7dialogremoveemailbutton[state='down'] {
background-image:url(images/sdk/dlgremove_over.png);
}
.s7ecatalogviewer .s7emaildialog .s7dialogremoveemailbutton[state='disabled'] {
background-image:url(images/sdk/dlgremove_up.png);
}

```

The content being shared is displayed in the bottom of the dialog box body and includes a thumbnail, title, origin URL, and description. It is wrapped into a container that is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogbody .s7dialogcontent
```

CSS properties of the dialog box content

border	Container border.
--------	-------------------

padding	Inner padding.
---------	----------------

Example – to set up a bottom container to have a one pixel dotted border and no padding:

```
.s7ecatalogviewer .s7emaildialog .s7dialogbody .s7dialogcontent {
  border: 1px dotted #A0A0A0;
  padding: 0;
}
```

Thumbnail image is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogthumbnail
```

The background-image property is set by the component logic.

CSS properties of the dialog box thumbnail image

width	Thumbnail width.
height	Thumbnail height.
vertical-align	Vertical alignment thumbnail.
padding	Inner padding.

Example – to set up thumbnail to be 90 x 60 pixels, and top-aligned with ten pixels of padding:

```
.s7ecatalogviewer .s7emaildialog .s7dialogthumbnail {
  height: 60px;
  padding: 10px;
  vertical-align: top;
  width: 90px;
}
```

Content title, origin, and description are further grouped into a panel to the right of the content thumbnail. It is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialoginfopanel
```

CSS properties of the dialog box information panel

width	Panel width.
-------	--------------

Example – to set up a content information panel to be 300 pixels wide:

```
.s7ecatalogviewer .s7emaildialog .s7dialoginfopanel {
  width: 300px;
}
```

Content title is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogtitle
```

CSS properties of the dialog box title

margin	Outer margin.
--------	---------------

font-weight	Font weight.
font-size	Font size.
font-family	Font family.

Example – to set up a content title to use bold font and have a ten pixel margin:

```
.s7ecatalogviewer .s7emaildialog .s7dialogtitle {
  font-weight: bold;
  margin: 10px;
}
```

Content origin is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogorigin
```

CSS properties of the dialog box content origin

margin	Outer margin.
font-weight	Font weight.
font-size	Font size.
font-family	Font family.

Example – to set up content origin to have a ten pixel margin:

```
.s7ecatalogviewer .s7emaildialog .s7dialogorigin {
  margin: 10px;
}
```

Content description is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogdescription
```

CSS properties of the dialog box content description

margin	Outer margin.
font-weight	Font weight.
font-size	Font size.
font-family	Font family.

Example – to set up a content description to have a ten pixel margin and use a nine point font:

```
.s7ecatalogviewer .s7emaildialog .s7dialogdescription {
  font-size: 9pt;
  margin: 10px;
}
```


When a user enters incorrect input data and inline validation fails, or when the dialog box needs to render an error or a confirmation message when the form is submitted, a message is displayed in the top of the dialog box body. It is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogerrormessage
```

CSS properties of the dialog box error message

background-image	Error icon. Default is an exclamation mark.
background-position	Error icon position inside the message area.
color	Message text color.
font-weight	Font weight.
font-size	Font size.
font-family	Font family.
line-height	Text height inside the message. Affects vertical alignment.
padding	Inner padding.



Note: This message supports the *state* attribute selector with the following possible values: *verifyerror*, *senderror*, and *sendsuccess*. *verifyerror* is set when a message is displayed due to an inline input validation failure; *senderror* is set when a backend email service reports an error; *sendsuccess* is set when email is sent successfully. This way it is possible to style the message differently depending on the dialog box state.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a message to use a ten point bold font, have 25 pixels line height, 20 pixels padding on the left, use an exclamation mark icon, red text in case of an error, and no icon and green text in case of success:

```
.s7ecatalogviewer .s7emaildialog .s7dialogerrormessage[state="verifyerror"] {
    background-image: url("images/sdk/dlgerrimg.png");
    color: #FF0000;
}
.s7ecatalogviewer .s7emaildialog .s7dialogerrormessage[state="senderror"] {
    background-image: url("images/sdk/dlgerrimg.png");
    color: #FF0000;
}
.s7ecatalogviewer .s7emaildialog .s7dialogerrormessage[state="sendsuccess"] {
    background-image: none;
    color: #00B200;
}
.s7ecatalogviewer .s7emaildialog .s7dialogerrormessage {
    background-position: left center;
    font-size: 10pt;
    font-weight: bold;
    line-height: 25px;
    padding-left: 20px;
}
```

If vertical scrolling is needed, the scroll bar is rendered in the panel near the right edge of the dialog, which is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7dialogscrollpanel
```

CSS properties of the dialog box scroll panel

width	Scroll panel width.
-------	---------------------

Example – to set up a scroll panel to be 44 pixels wide:

```
.s7ecatalogviewer .s7emaildialog .s7dialogscrollpanel {
    width: 44px;
}
```

The appearance of the scroll bar area is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7scrollbar
```

CSS properties of the scroll bar

width	The scroll bar width.
top	The vertical scroll bar offset from the top of the scroll panel.
bottom	The vertical scroll bar offset from the bottom of the scroll panel.
right	The horizontal scroll bar offset from the right edge of the scroll panel.

Example – to set up a scroll bar that is 28 pixels wide, an eight pixel margin from the top, right, and bottom of the scroll panel:

```
.s7ecatalogviewer .s7emaildialog .s7scrollbar {
    bottom: 8px;
    right: 8px;
    top: 8px;
    width: 28px;
}
```

Scroll bar track is the area between the top and bottom scroll buttons. The component automatically sets the position and height of the track. The track is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrolltrack
```

CSS properties of the scroll track

width	The track width.
background-color	The track background color.

Example – to set up a scroll bar track that is 28 pixels wide and has a gray background:

```
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrolltrack {
    width: 28px;
    background-color: #B2B2B2;
}
```

Scroll bar thumb moves vertically within a scroll track area. Its vertical position is fully controlled by the component logic, however the thumb height does not dynamically change depending on the amount of content. You can configure the thumb height and other aspects with the following CSS class selector:

```
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrollthumb
```

CSS properties of the scroll bar thumb

width	The thumb width.
height	The thumb height.
padding-top	The vertical padding between the top of the track.
padding-bottom	The vertical padding between the bottom of the track.
background-image	The image that is displayed for a given thumb state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: Thumb supports the *state* attribute selector, which can be used to apply different skins to different thumb states: *up*, *down*, *over*, and *disabled*.

Example – to set up scroll bar thumb that is 28 x 45 pixels, has a ten pixel margin on the top and the bottom, and has different artwork for each state:

```
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrollthumb {
  height: 45px;
  padding-bottom: 10px;
  padding-top: 10px;
  width: 28px;
}
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrollthumb[state="up"] {
  background-image:url("images/sdk/scrollbar_thumb_up.png");
}
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrollthumb[state="down"] {
  background-image:url("images/sdk/scrollbar_thumb_down.png");
}
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrollthumb[state="over"] {
  background-image:url("images/sdk/scrollbar_thumb_over.png");
}
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrollthumb[state="disabled"] {
  background-image:url("images/sdk/scrollbar_thumb_disabled.png");
}
```

The appearance of the top and bottom scroll buttons is controlled with the following CSS class selectors:

```
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrollupbutton
```

```
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton
```

It is not possible to position scroll buttons using CSS *top*, *left*, *bottom*, and *right* properties. Instead, the viewer logic automatically positions them.

CSS properties of the top and bottom scroll buttons

width	The button width.
height	The button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: These buttons support the *state* attribute selector, which can be used to apply different skins to different button states: *up*, *down*, *over*, and *disabled*.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up scroll buttons that are 28 x 32 pixels and have different artwork for each state:

```
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrollupbutton {
  width:28px;
  height:32px;
}
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrollupbutton[state='up'] {
  background-image:url(images/sdk/scroll_up_up.png);
}
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrollupbutton[state='over'] {
  background-image:url(images/sdk/scroll_up_over.png);
}
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrollupbutton[state='down'] {
  background-image:url(images/sdk/scroll_up_down.png);
}
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrollupbutton[state='disabled'] {
  background-image:url(images/sdk/scroll_up_disabled.png);
}
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton {
  width:28px;
  height:32px;
}
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton[state='up'] {
  background-image:url(images/sdk/scroll_down_up.png);
}
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton[state='over'] {
  background-image:url(images/sdk/scroll_down_over.png);
}
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton[state='down'] {
  background-image:url(images/sdk/scroll_down_down.png);
}
.s7ecatalogviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton[state='disabled'] {
  background-image:url(images/sdk/scroll_down_disabled.png);
}
```

Embed share

Embed share tool consists of a button added to the Social share panel and the modal dialog box that displays when the tool is activated. The position of the button is fully managed by the Social share tool.

The appearance of the embed share button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embedshare
```

CSS properties of the embed share tool

width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a embed share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```
.s7ecatalogviewer .s7embedshare {
  width:28px;
  height:28px;
}
.s7ecatalogviewer .s7embedshare[state='up'] {
background-image:url(images/v2/EmbedShare_dark_up.png);
}
.s7ecatalogviewer .s7embedshare[state='over'] {
background-image:url(images/v2/EmbedShare_dark_over.png);
}
.s7ecatalogviewer .s7embedshare[state='down'] {
background-image:url(images/v2/EmbedShare_dark_down.png);
}
.s7ecatalogviewer .s7embedshare[state='disabled'] {
background-image:url(images/v2/EmbedShare_dark_disabled.png);
}
```

The background overlay that covers the web page when the dialog box is active is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7backoverlay
```

CSS properties of the background overlay

opacity	Background overlay opacity.
background-color	Background overlay color.

Example – to set up a background overlay to be gray with 70% opacity:

```
.s7ecatalogviewer .s7embeddialog .s7backoverlay {
  opacity:0.7;
  background-color:#222222;
}
```

By default the modal dialog box is displayed centered in the screen on desktop systems and takes the entire web page area on touch devices. In all cases, the positioning and sizing of the dialog box is managed by the component. The dialog box is controlled with the following CSS class selector:

```
.s7embeddialog .s7dialog
```

CSS properties of the dialog box

border-radius	Dialog box border radius, in case the dialog box does not take the entire browser.
background-color	Dialog box background color.
width	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).
height	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).

Example – to set up the dialog box to use the entire browser window and have a white background on touch devices:

```
.s7ecatalogviewer .s7touchinput .s7embeddialog .s7dialog {
  width:100%;
  height:100%;
  background-color: #ffffff;
}
```

Dialog box header consists of an icon, a title text, and a close button. The header container is controlled with

```
.s7ecatalogviewer .s7embeddialog .s7dialogheader
```

CSS properties of the dialog box header

padding	Inner padding for header content.
---------	-----------------------------------

The icon and the title text are wrapped into an additional container controlled with

```
.s7ecatalogviewer .s7embeddialog .s7dialogheader .s7dialogline
```

CSS properties of the dialog line

padding	Inner padding for the header icon and title
---------	---

Header icon is controlled with the following CSS class selector

```
.s7ecatalogviewer .s7embeddialog .s7dialogheadericon
```

CSS properties of the dialog box header icon

width	Icon width.
height	Icon height.
background-image	Icon image.

background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
---------------------	--

Header title is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialogheadertext
```

CSS properties of the dialog box header text

font-weight	Font weight.
font-size	Font height.
font-family	Font family.
padding	Internal text padding.

Close button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7closebutton
```

CSS properties of the close button

top	Vertical button position relative to header container.
right	Horizontal button position relative to header container.
width	Button width.
height	Button height.
padding	Inner padding of button.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up dialog header with padding, 24 x 14 pixels icon, bold 16 point title, and 28 x 28 pixels close button, positioned two pixels from the top, and two pixels from the right of dialog container:

```
.s7ecatalogviewer .s7embeddialog .s7dialogheader {
  padding: 10px;
}
.s7ecatalogviewer .s7embeddialog .s7dialogheader .s7dialogline {
```

```
padding: 10px 10px 2px;
}
.s7ecatalogviewer .s7embeddialog .s7dialogheadericon {
    background-image: url("images/sdk/dlgembed_cap.png");
    height: 14px;
    width: 24px;
}
.s7ecatalogviewer .s7embeddialog .s7dialogheadertext {
    font-size: 16pt;
    font-weight: bold;
    padding-left: 16px;
}
.s7ecatalogviewer .s7embeddialog .s7closebutton {
    top: 2px;
    right: 2px;
    padding: 8px;
    width: 28px;
    height: 28px;
}
.s7ecatalogviewer .s7embeddialog .s7closebutton[state='up'] {
    background-image: url(images/sdk/close_up.png);
}
.s7ecatalogviewer .s7embeddialog .s7closebutton[state='over'] {
    background-image: url(images/sdk/close_over.png);
}
.s7ecatalogviewer .s7embeddialog .s7closebutton[state='down'] {
    background-image: url(images/sdk/close_down.png);
}
.s7ecatalogviewer .s7embeddialog .s7closebutton[state='disabled'] {
    background-image: url(images/sdk/close_disabled.png);
}
```

Dialog footer consists of "cancel" button. The footer container is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialogfooter
```

CSS properties of the dialog box footer

border	Border that you can use to visually separate the footer from the rest of the dialog box.
--------	--

The footer has an inner container which keeps the button. It is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialogbuttoncontainer
```

CSS properties of the dialog box button container

padding	Inner padding between the footer and the button.
---------	--

Select All button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialogactionbutton
```

The button is only available on desktop systems.

CSS properties of the Select All button

width	Button width.
height	Button height.
color	Button text color for each state.

background-color	Button background color for each state.
------------------	---



Note: The Select All button supports the `state` attribute selector, which can be used to apply different skins to different button states.

Cancel button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialogcancelbutton
```

CSS properties of the dialog box cancel button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

In addition, both buttons share the same common CSS class which can contain CSS settings that are the same for other dialog box buttons:

```
.s7ecatalogviewer .s7embeddialog .s7dialogfooter .s7button
```

CSS properties of the button

font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.
line-height	Text height inside the button. Affects vertical alignment.
box-shadow	Drop shadow.
margin-right	Right button margin.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box footer with a 64 x 34 Cancel button, an 82 x 34 Select All button, and having a text color and background color that is different for each button state:

```
.s7ecatalogviewer .s7embeddialog .s7dialogfooter {
  border-top: 1px solid #909090;
}
.s7ecatalogviewer .s7embeddialog .s7dialogbuttoncontainer {
```

```
padding-bottom: 6px;
padding-top: 10px;
}
.s7ecatalogviewer .s7embeddialog .s7dialogfooter .s7button {
  box-shadow: 1px 1px 1px #999999;
  color: #FFFFFF;
  font-size: 9pt;
  font-weight: bold;
  line-height: 34px;
  margin-right: 10px;
}
.s7ecatalogviewer .s7embeddialog .s7dialogcancelbutton {
  width:64px;
  height:34px;
}
.s7ecatalogviewer .s7embeddialog .s7dialogcancelbutton[state='up'] {
  background-color:#666666;
  color:#dddddd;
}
.s7ecatalogviewer .s7embeddialog .s7dialogcancelbutton[state='down'] {
  background-color:#555555;
  color:#ffffff;
}
.s7ecatalogviewer .s7embeddialog .s7dialogcancelbutton[state='over'] {
  background-color:#555555;
  color:#ffffff;
}
.s7ecatalogviewer .s7embeddialog .s7dialogcancelbutton[state='disabled'] {
  background-color:#b2b2b2;
  color:#dddddd;
}
.s7ecatalogviewer .s7embeddialog .s7dialogactionbutton {
  width:82px;
  height:34px;
}
.s7ecatalogviewer .s7embeddialog .s7dialogactionbutton[state='up'] {
  background-color:#333333;
  color:#dddddd;
}
.s7ecatalogviewer .s7embeddialog .s7dialogactionbutton[state='down'] {
  background-color:#222222;
  color:#cccccc;
}
.s7ecatalogviewer .s7embeddialog .s7dialogactionbutton[state='over'] {
  background-color:#222222;
  color:#cccccc;
}
.s7ecatalogviewer .s7embeddialog .s7dialogactionbutton[state='disabled'] {
  background-color:#b2b2b2;
  color:#dddddd;
}
}
```

The main dialog area (between the header and the footer) contains scrollable dialog content and scroll panel on the right. In all cases, the component manages the width of this area, it is not possible to set it in CSS. Main dialog area is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialogviewarea
```

CSS properties of the dialog box viewing area

height	The height of the main dialog box area. It should be specified only when the dialog box works in desktop mode. It is not applicable when the dialog box is sized to occupy the entire browser window.
background-color	The background color of the main dialog box area.

margin	Outer margin.
--------	---------------

Example – to set up a main dialog box area to be 300 pixels height, have a ten pixel margin, and use a white background:

```
.s7ecatalogviewer .s7embeddialog .s7dialogviewarea {
  background-color:#ffffff;
  margin:10px;
  height:300px;
}
```

All form content (like labels and input fields) resides inside a container controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialogbody
```

If the height of this container appears to be bigger than the main dialog box area, a vertical scroll is enabled automatically by the component.

CSS properties of the dialog box body

padding	Inner padding.
---------	----------------

Example – to set up form content to have ten pixel padding:

```
.s7ecatalogviewer .s7embeddialog .s7dialogbody {
  padding: 10px;
}
```

All static labels in the dialog box form are controlled with

```
.s7ecatalogviewer .s7embeddialog .s7dialoglabel
```

This class is not suitable for controlling the label size or position because you can apply it to texts in various places of the form user interface.

CSS properties of the dialog box label.

font-weight	Label font weight.
font-size	Label font size.
font-family	Label font family.
color	Label text color.

Dialog box labels can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up all labels to be gray, bold with a nine pixel font:

```
.s7ecatalogviewer .s7embeddialog .s7dialoglabel {
  color: #666666;
  font-size: 9pt;
  font-weight: bold;
}
```

The size of the text copy displayed on top of the embed code is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialoginputwide
```

CSS properties of the dialog box input wide field

width	Input field width.
padding	Inner padding.

Example – to set text copy to be 430 pixels wide and have a ten pixel padding in the bottom:

```
.s7ecatalogviewer .s7embeddialog .s7dialoginputwide {
  padding-bottom: 10px;
  width: 430px;
}
```

The embed code is wrapped into container and controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

width	The width of the embed code container.
border	Border around the embed code container.
padding	Inner padding.

Example – to set a one pixel grey border around embed code text, make it 430 pixels wide, and have a ten pixel padding:

```
.s7ecatalogviewer .s7embeddialog .s7dialoginputcontainer {
  border: 1px solid #CCCCCC;
  padding: 10px;
  width: 430px;
}
```

The actual embed code text is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

word-wrap	Word wrapping style.
-----------	----------------------

Example – to set up embed code to use break-word word wrapping:

```
.s7ecatalogviewer .s7embeddialog .s7dialogmessage {
  word-wrap: break-word;
}
```

Embed size label and drop-down are located in the bottom of the dialog box and put into a container controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialogembedsizepanel
```

CSS properties of the dialog box embed size panel

padding	Inner padding.
---------	----------------

Example – to set up an embed size panel to have ten pixels of padding:

```
.s7ecatalogviewer .s7embeddialog .s7dialogembedsizepanel {
    padding: 10px;
}
```

The size and alignment of the embed size label is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialogembedsizelabel
```

CSS properties of the dialog box embed size panel

vertical-align	Vertical label alignment.
width	Label width.

Example – to set the embed size label to be top-aligned and 80 pixels wide:

```
.s7ecatalogviewer .s7embeddialog .s7dialogembedsizelabel {
    vertical-align: top;
    width: 80px;
}
```

The width of the embed size combo box is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7combobox
```

CSS properties of the combo box

width	Combo box width.
-------	------------------



Note: The combo box supports the `expanded` attribute selector with possible values of `true` and `false`. `true` is used when combo box displays one of pre-defined embed sizes, thus should take all available width. `false` is used when custom size option is selected in the combo box, so it should shrink to allow space for custom width and height input fields.

Example – to set the embed size combo box to be 300 pixels wide when showing a pre-defined item and 110 pixels wide when showing a custom size:

```
.s7ecatalogviewer .s7embeddialog .s7combobox[expanded="true"] {
    width: 300px;
}
.s7ecatalogviewer .s7embeddialog .s7combobox[expanded="false"] {
    width: 110px;
}
```

The height of the combo box text is defined by a special inner element and is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7combobox .s7comboboxtext
```

CSS properties of the combo box text

height	Combo box text height.
--------	------------------------

Example – to set embed size combo box text height to 40 pixels:

```
.s7ecatalogviewer .s7embeddialog .s7combobox .s7comboboxtext {
    height: 40px;
}
```

The combo box has a "drop down" button on the right and it is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7combobox .s7comboboxbutton
```

CSS properties of the combo box button

top	Vertical button position inside the combo box.
right	Horizontal button position inside the combo box.
width	Button width.
height	Button height.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

Example – to set a drop-down button to 28 x 28 pixels and have a separate image for each state:

```
.s7ecatalogviewer .s7embeddialog .s7combobox .s7comboboxbutton {
  width:28px;
  height:28px;
}
.s7ecatalogviewer .s7embeddialog .s7combobox .s7comboboxbutton[state='up'] {
  background-image:url(images/sdk/cboxbtndn_up.png);
}
.s7ecatalogviewer .s7embeddialog .s7combobox .s7comboboxbutton[state='over'] {
  background-image:url(images/sdk/cboxbtndn_over.png);
}
.s7ecatalogviewer .s7embeddialog .s7combobox .s7comboboxbutton[state='down'] {
  background-image:url(images/sdk/cboxbtndn_over.png);
}
.s7ecatalogviewer .s7embeddialog .s7combobox .s7comboboxbutton[state='disabled'] {
  background-image:url(images/sdk/cboxbtndn_up.png);
}
```

The panel with the list of embed sizes displayed when combo box is opened is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7comboboxdropdown
```

The size and position of the panel is controlled by the component. It is not possible to change it through CSS.

CSS properties of the combo box drop-down

border	Panel border.
--------	---------------

Example – to set the combo box panel to have a one pixel grey border:

```
.s7ecatalogviewer .s7embeddialog .s7comboboxdropdown {
  border: 1px solid #CCCCCC;
}
```

A single item in a drop-down panel that is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dropdownitemanchor
```

CSS properties of the drop-down item anchor

background-color	Item background.
------------------	------------------

Example – to set the combo box panel item to have a white background:

```
.s7ecatalogviewer .s7embeddialog .s7dropdownitemanchor {
    background-color: #FFFFFF;
}
```

A check mark displayed to the left of the selected item inside the combo box panel that is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7checkmark
```

CSS properties of the check mark box

width	Icon width.
height	Icon height.
background-image	Item image.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .

Example – to set the check mark icon to 25 x 25 pixels:

```
.s7ecatalogviewer .s7embeddialog .s7checkmark {
    background-image: url("images/sdk/cboxchecked.png");
    height: 25px;
    width: 25px;
}
```

When "Custom Size" option is selected in the embed size combo box the dialog box displays two extra input fields to the right to allow the user to enter a custom embed size. Those fields are wrapped in a container that is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialogcustomsizepanel
```

CSS properties of the dialog box custom size panel

left	Distance from the embed size combo box.
------	---

Example – to set custom size input fields panel to be 20 pixels to the right of the combo box:

```
.s7ecatalogviewer .s7embeddialog .s7dialogcustomsizepanel {
    left: 20px;
}
```

Each custom size input field is wrapped in a container that renders a border and sets the margin between the fields. It is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialogcustomsize
```

CSS properties of the dialog box custom size

border	Border around the input field.
width	Input field width.
margin	Input field margin.
padding	Input field padding.

Example – to set the custom size input fields to have a one pixel grey border, margin, padding and be 70 pixels wide:

```
.s7ecatalogviewer .s7embeddialog .s7dialogcustomsize {
  border: 1px solid #CCCCCC;
  margin-right: 20px;
  padding-left: 2px;
  padding-right: 2px;
  width: 70px;
}
```

If vertical scrolling is needed, the scroll bar is rendered in the panel near the right edge of the dialog box, which is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7dialogscrollpanel
```

CSS properties of the dialog box scroll panel

width	Scroll panel width.
-------	---------------------

Example – to set up a scroll panel to be 44 pixels wide

```
.s7ecatalogviewer .s7embeddialog .s7dialogscrollpanel {
  width: 44px;
}
```

The appearance of the scroll bar area is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7scrollbar
```

CSS properties of the scroll bar

width	Scroll bar width.
top	The vertical scroll bar offset from the top of the scroll panel.
bottom	The vertical scroll bar offset from the bottom of the scroll panel.
right	The horizontal scroll bar offset from the right edge of the scroll panel.

Example – to set up a scroll bar that is 28 pixels wide and has an eight pixel margin from the top, right, and bottom of the scroll panel:

```
.s7ecatalogviewer .s7embeddialog .s7scrollbar {
  bottom: 8px;
  right: 8px;
  top: 8px;
  width: 28px;
}
```

Scroll bar track is the area between the top and bottom scroll buttons. The component automatically sets the position and height of the track. The track is controlled with the following CSS class selector

```
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrolltrack
```

CSS properties of the scroll bar track

width	Track width.
background-color	Track background color.

Example – to set up a scroll bar track that is 28 pixels wide and has a grey background:

```
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrolltrack {
width:28px;
background-color: #B2B2B2;
}
```

The scroll bar thumb moves vertically within a scroll track area. Its vertical position is fully controlled by the component logic. However, thumb height does not dynamically change depending on the amount of content. The thumb height and other aspects can be configured with the following CSS class selector:

```
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrollthumb
```

CSS properties of the scroll bar thumb

width	Thumb width.
height	Thumb height.
padding-top	The vertical padding between the top of the track.
padding-bottom	The vertical padding between the bottom of the track.
background-image	The image displayed for a given thumb state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: Thumb supports the *state* attribute selector, which can be used to apply different skins to different thumb states: up, down, over, and disabled.

Example – to set up a scroll bar thumb that is 28 x 45 pixels, has a ten pixel margin on the top and bottom, and has different artwork for each state:

```
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrollthumb {
  height: 45px;
  padding-bottom: 10px;
  padding-top: 10px;
  width: 28px;
}
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrollthumb[state="up"] {
  background-image:url("images/sdk/scrollbar_thumb_up.png");
}
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrollthumb[state="down"] {
  background-image:url("images/sdk/scrollbar_thumb_down.png");
}
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrollthumb[state="over"] {
  background-image:url("images/sdk/scrollbar_thumb_over.png");
}
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrollthumb[state="disabled"] {
  background-image:url("images/sdk/scrollbar_thumb_disabled.png");
}
```

The appearance of the top and bottom scroll buttons is controlled with the following CSS class selectors:

```
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrollupbutton
```

```
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton
```

It is not possible to position scroll buttons using CSS top, left, bottom, and right properties. Instead, the viewer logic automatically positions them.

CSS properties of the top and bottom scroll buttons

width	Button width.
height	Button height.
background-image	The image displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: These buttons support the *state* attribute selector, which can be used to apply different skins to different button states: *up*, *down*, *over*, and *disabled*.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up scroll buttons that are 28 x 32 pixels and have different artwork for each state:

```
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrollupbutton {
  width:28px;
  height:32px;
}
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrollupbutton[state='up'] {
  background-image:url(images/sdk/scroll_up_up.png);
}
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrollupbutton[state='over'] {
  background-image:url(images/sdk/scroll_up_over.png);
}
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrollupbutton[state='down'] {
```

```

background-image:url(images/sdk/scroll_up_down.png);
}
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrollupbutton[state='disabled'] {
background-image:url(images/sdk/scroll_up_disabled.png);
}
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton {
width:28px;
height:32px;
}
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton[state='up'] {
background-image:url(images/sdk/scroll_down_up.png);
}
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton[state='over'] {
background-image:url(images/sdk/scroll_down_over.png);
}
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton[state='down'] {
background-image:url(images/sdk/scroll_down_down.png);
}
.s7ecatalogviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton[state='disabled'] {
background-image:url(images/sdk/scroll_down_disabled.png);
}
}

```

Facebook share

Facebook share tool consists of a button added to the Social share panel. When the button is clicked the user is redirected to a sharing dialog box that is provided by a social service. The position of the button is fully managed by the Social share tool.

The appearance of the Facebook share button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7facebookshare
```

CSS properties of the Facebook share tool

width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Facebook share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```

.s7ecatalogviewer .s7facebookshare {
width:28px;
height:28px;
}
.s7ecatalogviewer .s7facebookshare[state='up'] {
background-image:url(images/v2/FacebookShare_dark_up.png);
}
.s7ecatalogviewer .s7facebookshare[state='over'] {

```

```
background-image:url(images/v2/FacebookShare_dark_over.png);
}
.s7ecatalogviewer .s7facebookshare[state='down'] {
background-image:url(images/v2/FacebookShare_dark_down.png);
}
.s7ecatalogviewer .s7facebookshare[state='disabled'] {
background-image:url(images/v2/FacebookShare_dark_disabled.png);
}
```

Favorites menu

The Favorites menu drop-down list appears in the control bar. It consists of a button and a panel that expands when a user clicks or taps on a button. The panel contains individual Favorites tools.

The position and size of the Favorites menu in the viewer user interface is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7favoritesmenu
```

CSS properties of the Favorites menu button

margin-top	The offset from the top of the control bar.
margin-left	The distance to the next button on the left, or the left side of the control bar if this is the first button in a row.
width	Width of the button.
height	Height of the button.

Example – set up a Favorites menu that is positioned four pixels from the top of the control bar and ten pixels from the closest button to the left and sized 28 x 28 pixels.

```
.s7ecatalogviewer .s7favoritesmenu {
margin-top: 4px;
margin-left: 10px;
width:28px;
height:28px;
}
```

The appearance of the Favorites menu button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7favoritesmenu .s7favoritesbutton
```

CSS properties of the Favorites button

background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – set up a Favorites menu button that displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7favoritesmenu .s7favoritesbutton[state='up'] {
background-image:url(images/v2/FavoritesMenu_dark_up.png);
}
.s7ecatalogviewer .s7favoritesmenu .s7favoritesbutton[state='over'] {
background-image:url(images/v2/FavoritesMenu_dark_over.png);
}
.s7ecatalogviewer .s7favoritesmenu .s7favoritesbutton[state='down'] {
background-image:url(images/v2/FavoritesMenu_dark_down.png);
}
.s7ecatalogviewer .s7favoritesmenu .s7favoritesbutton[state='disabled'] {
background-image:url(images/v2/FavoritesMenu_dark_disabled.png);
}
```

The appearance of the panel that contains individual Favorites icons is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7favoritesmenu .s7favoritesmenupanel
```

CSS properties of the Favorites menu panel

background-color	The background color of the panel.
------------------	------------------------------------

Example – set up a panel to have a transparent color.

```
.s7ecatalogviewer .s7favoritesmenu .s7favoritesmenupanel {
background-color: transparent;
}
```

Favorites effect

The viewer displays Favorites icons over the main view in places where it was originally added by the user.

The appearance of the Favorite icon is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7favoriteseffect .s7icon
```

CSS properties of the Favorite icon

background-image	The image that is displayed for the icon.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
width	Width of the icon.
height	Height of the icon.

Example – set up a 36 x 36 pixels Favorites icon.

```
.s7ecatalogviewer .s7favoriteseffect .s7icon {
height: 36px;
width: 36px;
background-image: url(images/v2/FavoriteEffect_dark_up.png);
}
```

On desktop systems, the component supports the `cursor` attribute selector which you can apply to the `.s7favoriteseffect` class and controls the type of the cursor based on the selected user action. The following `cursor` values are supported:

<code>mode_add</code>	Displayed user is adding a new Favorite icon.
<code>mode_remove</code>	Displayed user is removing an existing Favorite icon.
<code>mode_view</code>	Displayed in normal operation mode when Favorites editing is not active.

Example – have different mouse cursors for each type of component state.

```
.s7ecatalogviewer .s7favoriteseffect[cursor="mode_add"] {
  cursor: crosshair;
}
.s7ecatalogviewer .s7favoriteseffect[cursor="mode_remove"] {
  cursor: not-allowed;
}
.s7ecatalogviewer .s7favoriteseffect[cursor="mode_view"] {
  cursor: auto;
}
```

Favorites view

Favorites view consist of a column of thumbnail images.

The appearance of favorites view container is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7favoritesview
```

The position and the height of the Favorites view is managed by the view; in CSS it is only possible to define the width.

CSS properties of the Favorites view

<code>background-color</code>	Background color of the Favorites view.
<code>width</code>	Width of the view.

Example – to set up a Favorites view that is 100 pixels wide with a semi-transparent grey background.

```
.s7ecatalogviewer .s7favoritesview {
  width: 100px;
  background-color: rgba(221, 221, 221, 0.5);
}
```

The spacing between Favorites thumbnails is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7favoritesview .s7thumbcell
```

CSS properties of the Favorites thumbnails

<code>margin</code>	The size of the vertical margin around each thumbnail. The actual thumbnail spacing is equal to the sum of the top and bottom margin that is set for <code>.s7thumbcell</code> .
---------------------	--

Example – to set up 10 pixel spacing.

```
.s7ecatalogviewer .s7favoritesview .s7thumbcell {
  margin: 5px;
}
```

The appearance of individual thumbnail is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7favoritesview .s7thumb
```

CSS properties of the Favorites thumbnails

width	Width of the thumbnail.
height	Height of the thumbnail.
border	Border of the thumbnail.



Note: Thumbnail supports the `state` attribute selector, which can be used to apply different skins to different thumbnail states. In particular, `state="selected"` corresponds to the thumbnail recently selected by the user. `state="default"` corresponds to the rest of the thumbnails. And `state="over"` is used on mouse hover.

Example – to set up thumbnails that are 75 x 75 pixels, have a light grey default border, and a dark grey selected border.

```
.s7ecatalogviewer .s7favoritesview .s7thumb {
  width: 75px;
  height: 75px;
}
.s7ecatalogviewer .s7favoritesview .s7thumb[state="default"] {
  border: 1px solid #dddddd;
}
.s7ecatalogviewer .s7favoritesview .s7thumb[state="selected"] {
  border: 1px solid #666666;
}
```

The appearance of the thumbnail label is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7favoritesview .s7label
```

CSS properties of the Favorites label

font-family	Font name.
font-size	Font size.

Example – to set up labels with a 14 pixel Helvetica font.

```
.s7ecatalogviewer .s7favoritesview .s7label {
  font-family: Helvetica, sans-serif;
  font-size: 14px;
}
```

First page button

Clicking or tapping on this button brings the user to the first page in the catalog. This button appears in the main control bar on desktop systems and tablets; on mobile phones it is added to a secondary control bar. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7firstpagebutton .s7panleftbutton
```

CSS property	Description
top	Position from the top border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
right	Position from the right border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
left	Position from the left border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
bottom	Position from the bottom border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a first page button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 220 pixels from the left edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7firstpagebutton .s7panleftbutton {
bottom:4px;
left:220px;
width:32px;
height:32px;
}
.s7ecatalogviewer .s7firstpagebutton .s7panleftbutton [state='up'] {
background-image:url(images/v2/FirstPageButton_dark_up.png);
}
.s7ecatalogviewer .s7firstpagebutton .s7panleftbutton [state='over'] {
background-image:url(images/v2/FirstPageButton_dark_over.png);
}
.s7ecatalogviewer .s7firstpagebutton .s7panleftbutton [state='down'] {
background-image:url(images/v2/FirstPageButton_dark_down.png);
}
.s7ecatalogviewer .s7firstpagebutton .s7panleftbutton [state='disabled'] {
```



```
background-image:url(images/v2/FirstPageButton_dark_disabled.png);
}
```

Full screen button

Causes the viewer to enter or exit full screen mode when clicked by the user. This button appears in the main control bar. This button is not displayed if the viewer works in pop-up mode and the system does not support native full screen. You can size, skin, and position the button by CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7fullscreenbutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.
left	Position from the left border of the main control bar, including padding.
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the "full screen" state and *selected='false'* corresponds to the "normal" state.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a full screen button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 5 pixels from the right edge of the main control bar, and displays a different image for each of the four different button states when selected or not selected.

```
.s7ecatalogviewer .s7fullscreenbutton {
bottom:4px;
right:5px;
width:28px;
height:28px;
}
.s7ecatalogviewer .s7fullscreenbutton [selected='false'][state='up'] {
background-image:url(images/enterFullBtn_up.png);
}
```

```
}
.s7ecatalogviewer .s7fullscreenbutton [selected='false'][state='over'] {
background-image:url(images/enterFullBtn_over.png);
}
.s7ecatalogviewer .s7fullscreenbutton [selected='false'][state='down'] {
background-image:url(images/enterFullBtn_down.png);
}
.s7ecatalogviewer .s7fullscreenbutton [selected='false'][state='disabled'] {
background-image:url(images/enterFullBtn_disabled.png);
}
.s7ecatalogviewer .s7fullscreenbutton [selected='true'][state='up'] {
background-image:url(images/exitFullBtn_up.png);
}
.s7ecatalogviewer .s7fullscreenbutton [selected='true'][state='over'] {
background-image:url(images/exitFullBtn_over.png);
}
.s7ecatalogviewer .s7fullscreenbutton [selected='true'][state='down'] {
background-image:url(images/exitFullBtn_down.png);
}
.s7ecatalogviewer .s7fullscreenbutton [selected='true'][state='disabled'] {
background-image:url(images/exitFullBtn_disabled.png); }
}
```

Icon effect


The zoom indicator is overlaid on the main view area. It is displayed when the image is in a reset state and it also depends on `iconeffect` parameter.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7pageview .s7iconeffect
```

CSS property	Description
background-image	Zoom indicator artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
width	Zoom indicator width in pixels.
height	Zoom indicator height in pixels.

 **Note:** Icon effect supports the `media-type` attribute selector, which you can use to apply different icon effects on different devices. In particular, `media-type='standard'` corresponds to desktop systems where mouse input is normally used and `media-type='multitouch'` corresponds to devices with touch input.

Example – to set up a 100 x 100 pixel zoom indicator with different art for desktop systems and touch devices.

```
.s7ecatalogviewer .s7pageview .s7iconeffect {
width: 100px;
height: 100px;
}
.s7ecatalogviewer .s7pageview .s7iconeffect[media-type='standard'] {
background-image:url(images/v2/IconEffect_zoom.png);
}
```

```
.s7ecatalogviewer .s7pageview .s7iconeffect[media-type='multitouch'] {
  background-image:url(images/v2/IconEffect_pinch.png);
}
```

Image map effect

Depending on the value of the `mode` parameter, the viewer displays image map icons over the main view in places where maps are originally authored in Scene7 Publishing System or renders exact regions that match the shape of original image maps.

CSS properties of the main viewer area

The appearance of the image map icon is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7imagemapeffect .s7icon
```



Note: The `s7mapoverlay` CSS class used to style image map icons in the past is now deprecated; use `s7icon` instead.

CSS property	Description
<code>background-image</code>	Image map icon artwork.
<code>background-position</code>	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
<code>width</code>	Image map icon width in pixels.
<code>height</code>	Image map icon height in pixels.



Note: Image map icon supports the `state` attribute selector, which you can use to apply different skins to the icon states of `default` and `active`.

Example – set up a 28 x 28 pixels image map icon that displays a different image for each of the two different icon states.

```
.s7ecatalogviewer .s7imagemapeffect .s7icon {
  height: 28px;
  width: 28px;
  background-image: url(images/v2/ImageMapEffect_dark_up.png);
}
.s7ecatalogviewer .s7imagemapeffect .s7icon[state="default"] {
  opacity: 0.5;
}
.s7ecatalogviewer .s7imagemapeffect .s7icon[state="active"] {
  opacity: 1;
}
```

See also [Image map support](#).

The appearance of the image map region is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7imagemapeffect .s7region
```

CSS property	Description
<code>background</code>	Image map region fill color.

CSS property	Description
	Specified in #RRGGBB, RGB(R,G,B), or RGBA(R,G,B,A) format.
background-color	Image map region fill color. Specified in #RRGGBB, RGB(R,G,B) or RGBA(R,G,B,A) format.
border	Image map region border style. Specified as <i>width solid color</i> , where <i>width</i> is expressed in pixels and <i>color</i> is set as #RRGGBB, RGB(R,G,B) or RGBA(R,G,B,A).

Example – set up a transparent image map region with 1 pixel black border :

```
.s7ecatalogviewer .s7imagemapeffect .s7region {
  border: 1px solid #000000;
  background: RGBA(0,0,0,0);
}
```

Info panel popup

Info Panel Popup displays in the middle of the viewer area when a user activates an image map that has a `rollover_key` property defined in Scene7 Publishing System, and if info panel feature is properly configured for the viewer.

Info panel background covers entire viewer area and is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7infopanelpopup .s7backoverlay
```

CSS property	Description
background-image	Info panel background fill.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .

Example – set up the info panel popup to use a semi-transparent black background.

```
.s7ecatalogviewer .s7infopanelpopup .s7backoverlay {
  background-color : rgba(0,0,0,0.75);
}
```

The info panel dialog is displayed by default in the middle of the viewer area. However it is possible to control its size, alignment, background, and border with the CSS class selector.

```
.s7ecatalogviewer .s7infopanelpopup .s7overlay
```

CSS property	Description
left	Horizontal position of the info panel dialog within viewer area panel background fill.

CSS property	Description
top	Vertical position of the info panel dialog within viewer area.
width	Dialog width.
height	Dialog height.
margin-left	Left margin of the info panel dialog, may be used for centering purposes.
margin-top	Top margin of the info panel dialog, may be used for centering purposes.
padding	Internal dialog padding.
background-color	Dialog background color.
border-radius	Dialog border radius.
box-shadow	Dialog shadow.

Example – set up 300 x 200 pixels info panel dialog that is centered in the viewer area; has 40 pixels padding at the top and 10 pixels padding on all other sides, a light gray background, and a 10 pixel border radius and drop shadow.

```
.s7ecatalogviewer .s7infopanelpopup .s7overlay {
  left: 50%;
  top: 50%;
  margin-left: -150px;
  margin-top: -100px;
  width: 300px;
  height: 200px;
  padding-top: 40px;
  padding-right: 10px;
  padding-bottom: 10px;
  padding-left: 10px;
  background-color:rgb(221,221,221);
  border-radius: 10px 10px 10px 10px;
  box-shadow: 0 0 5px rgba(0,0,0,0.25);
}
```

The Info Panel dialog has a close button, and clicking or tapping the button closes the dialog.

The appearance of this button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7infopanelpopup .s7closebutton
```

CSS property	Description
top	Position from the top border of the dialog.
right	Position from the right border of the dialog.

CSS property	Description
left	Position from the left border of the dialog.
bottom	Position from the bottom border of the dialog.
width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which you can use to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog close button that is 28 x 28 pixels, positioned 5 pixels from the top and right edge of the info panel dialog, and displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7infopanelpopup .s7closebutton {
  width: 28px;
  height: 28px;
  top: 5px;
  right: 5px;
}
.s7ecatalogviewer .s7infopanelpopup .s7closebutton[state="up"] {
  background-image:url(images/v2/InfoPanelPopup_CloseButton_dark_up.png);
}
.s7ecatalogviewer .s7infopanelpopup .s7closebutton[state="over"] {
  background-image:url(images/v2/InfoPanelPopup_CloseButton_dark_over.png);
}
.s7ecatalogviewer .s7infopanelpopup .s7closebutton[state="down"] {
  background-image:url(images/v2/InfoPanelPopup_CloseButton_dark_up.png);
}
.s7ecatalogviewer .s7infopanelpopup .s7closebutton[state="disabled"] {
  background-image:url(images/v2/InfoPanelPopup_CloseButton_dark_up.png);
}
```

Large next page button

Clicking or tapping on this button brings the user to the next page in the catalog. This button appears in the main control bar. This button is not displayed on mobile phones to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7ecatrightbutton .s7panrightbutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.
left	Position from the left border of the main control bar, including padding.
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a large next page button that is 56 x 56 pixels, vertically centered and anchored to the right viewer border, and displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7ecatrightbutton .s7panrightbutton {
bottom:50%;
margin-bottom:-28px;
right:0px;
width:56px;
height:56px;
}
.s7ecatalogviewer .s7ecatrightbutton .s7panrightbutton [state='up'] {
background-image:url(images/v2/RightButton_dark_up.png);
}
.s7ecatalogviewer .s7ecatrightbutton .s7panrightbutton [state='over'] {
background-image:url(images/v2/RightButton_dark_over.png);
}
.s7ecatalogviewer .s7ecatrightbutton .s7panrightbutton [state='down'] {
background-image:url(images/v2/RightButton_dark_down.png);
}
.s7ecatalogviewer .s7ecatrightbutton .s7panrightbutton [state='disabled'] {
background-image:url(images/v2/RightButton_dark_disabled.png);
}
```

Last page button

Clicking or tapping on this button brings the user to the last page in the catalog. This button appears in the main control bar on desktop systems and tablets; on mobile phones it is added to a secondary control bar. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7lastpagebutton .s7panleftbutton
```

CSS property	Description
top	Position from the top border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
right	Position from the right border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
left	Position from the left border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
bottom	Position from the bottom border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a last page button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 220 pixels from the left edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7lastpagebutton .s7panrightbutton {
bottom:4px;
right:220px;
width:32px;
height:32px;
}
.s7ecatalogviewer .s7lastpagebutton .s7panrightbutton [state='up'] {
background-image:url(images/v2/LastPageButton_dark_up.png);
}
.s7ecatalogviewer .s7lastpagebutton .s7panrightbutton [state='over'] {
background-image:url(images/v2/LastPageButton_dark_over.png);
}
.s7ecatalogviewer .s7lastpagebutton .s7panrightbutton [state='down'] {
background-image:url(images/v2/LastPageButton_dark_down.png);
}
.s7ecatalogviewer .s7lastpagebutton .s7panrightbutton [state='disabled'] {
background-image:url(images/v2/LastPageButton_dark_disabled.png);
}
```


Large previous page button

Clicking or tapping on this button brings the user to the previous page in the catalog. This button appears in the main control bar. This button is not displayed on mobile phones to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7ecatleftbutton .s7panleftbutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.
left	Position from the left border of the main control bar, including padding.
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a large previous page button that is 56 x 56 pixels, vertically centered and anchored to the left viewer border, and displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7ecatleftbutton .s7panleftbutton {
  bottom:50%;
  margin-bottom:-28px;
  left:0px;
  width:56px;
  height:56px;
}
.s7ecatalogviewer .s7ecatleftbutton .s7panleftbutton [state='up'] {
  background-image:url(images/v2/LeftButton_dark_up.png);
}
.s7ecatalogviewer .s7ecatleftbutton .s7panleftbutton [state='over'] {
  background-image:url(images/v2/LeftButton_dark_over.png);
}
.s7ecatalogviewer .s7ecatleftbutton .s7panleftbutton [state='down'] {
  background-image:url(images/v2/LeftButton_dark_down.png);
}
```

```

}
.s7ecatalogviewer .s7ecatleftbutton .s7panleftbutton [state='disabled'] {
background-image:url(images/v2/LeftButton_dark_disabled.png);
}

```

Link share

Link share tool consists of a button added to the Social share panel and the modal dialog box that displays when the tool is activated. The position of the button is fully managed by the Social share tool.

The appearance of the link share button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkshare
```

CSS properties of the link share tool

width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a link share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```

.s7ecatalogviewer .s7linkshare {
width:28px;
height:28px;
}
.s7ecatalogviewer .s7linkshare[state='up'] {
background-image:url(images/v2/LinkShare_dark_up.png);
}
.s7ecatalogviewer .s7linkshare[state='over'] {
background-image:url(images/v2/LinkShare_dark_over.png);
}
.s7ecatalogviewer .s7linkshare[state='down'] {
background-image:url(images/v2/LinkShare_dark_down.png);
}
.s7ecatalogviewer .s7linkshare[state='disabled'] {
background-image:url(images/v2/LinkShare_dark_disabled.png);
}

```

The background overlay that covers the web page when the dialog box is active is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkdialog .s7backoverlay
```

CSS properties of the background overlay

opacity	Background overlay opacity.
---------	-----------------------------

background-color	Background overlay color.
------------------	---------------------------

Example – to set up a background overlay to be gray with 70% opacity:

```
.s7ecatalogviewer .s7linkdialog .s7backoverlay {
  opacity:0.7;
  background-color:#222222;
}
```

By default the modal dialog box is displayed centered in the screen on desktop systems and takes the entire web page area on touch devices. In all cases, the positioning and sizing of the dialog box is managed by the component. The dialog box is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkdialog .s7dialog
```

CSS properties of the dialog box

border-radius	Dialog box border radius, in case the dialog box does not take the entire browser.
background-color	Dialog box background color.
width	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).
height	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).

Example – to set up the dialog box to use the entire browser window and have a white background on touch devices:

```
.s7ecatalogviewer .s7touchinput .s7linkdialog .s7dialog {
  width:100%;
  height:100%;
  background-color: #ffffff;
}
```

Dialog box header consists of an icon, a title text, and a close button. The header container is controlled with

```
.s7ecatalogviewer .s7linkdialog .s7dialogheader
```

CSS properties of the dialog box header

padding	Inner padding for header content.
---------	-----------------------------------

The icon and the title text are wrapped into an additional container controlled with

```
.s7ecatalogviewer .s7linkdialog .s7dialogheader .s7dialogline
```

CSS properties of the dialog line

padding	Inner padding for the header icon and title.
---------	--

Header icon is controlled with the following CSS class selector

```
.s7ecatalogviewer .s7linkdialog .s7dialogheadericon
```

CSS properties of the dialog box header icon

width	Icon width.
height	Icon height.
background-image	Icon image.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .

Header title is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkdialog .s7dialogheadertext
```

CSS properties of the dialog box header text

font-weight	Font weight.
font-size	Font height.
font-family	Font family.
padding	Internal text padding.

Close button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkdialog .s7closebutton
```

CSS properties of the close button

top	Vertical button position relative to header container.
right	Horizontal button position relative to header container.
width	Button width.
height	Button height.
padding	Inner padding of button.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The Close button tool tip and the dialog box title can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box header with padding, 22 x 12 pixels icon, bold 16 point title, and a 28 x 28 pixel Close button that is positioned two pixels from the top and two pixels from the right of the dialog box container:

```
.s7ecatalogviewer .s7linkdialog .s7dialogheader {
padding: 10px;
}
.s7ecatalogviewer .s7linkdialog .s7dialogheader .s7dialogline {
padding: 10px 10px 2px;
}
.s7ecatalogviewer .s7linkdialog .s7dialogheadericon {
background-image: url("images/sdk/dlglink_cap.png");
height: 12px;
width: 22px;
}
.s7ecatalogviewer .s7linkdialog .s7dialogheadertext {
font-size: 16pt;
font-weight: bold;
padding-left: 16px;
}
.s7ecatalogviewer .s7linkdialog .s7closebutton {
top: 2px;
right: 2px;
padding: 8px;
width: 28px;
height: 28px;
}
.s7ecatalogviewer .s7linkdialog .s7closebutton[state='up'] {
background-image: url(images/sdk/close_up.png);
}
.s7ecatalogviewer .s7linkdialog .s7closebutton[state='over'] {
background-image: url(images/sdk/close_over.png);
}
.s7ecatalogviewer .s7linkdialog .s7closebutton[state='down'] {
background-image: url(images/sdk/close_down.png);
}
.s7ecatalogviewer .s7linkdialog .s7closebutton[state='disabled'] {
background-image: url(images/sdk/close_disabled.png);
}
```

Dialog box footer consists of a Cancel button. The footer container is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkdialog .s7dialogfooter
```

CSS properties of the dialog box footer

border	Border that you can use to visually separate the footer from the rest of the dialog box.
--------	--

The footer has an inner container which keeps the button. It is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkdialog .s7dialogbuttoncontainer
```

CSS properties of the dialog box button container

padding	Inner padding between the footer and the button.
---------	--

Select All button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkdialog .s7dialogactionbutton
```

The button is only available on desktop systems.

CSS properties of the Select All button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: The Select All button supports the `state` attribute selector, which can be used to apply different skins to different button states.

Cancel button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkdialog .s7dialogcancelbutton
```

CSS properties of the dialog box cancel button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

In addition, both buttons share the same common CSS class which can contain CSS settings that are the same for other dialog box buttons:

```
.s7ecatalogviewer .s7linkdialog .s7dialogfooter .s7button
```

CSS properties of the button

font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.
line-height	Text height inside the button. Affects vertical alignment.
box-shadow	Drop shadow.
margin-right	Right button margin.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box footer with a 64 x 34 Cancel button, having text color and background color different for each button state:

```
.s7ecatalogviewer .s7linkdialog .s7dialogfooter {
  border-top: 1px solid #909090;
}
.s7ecatalogviewer .s7linkdialog .s7dialogbuttoncontainer {
  padding-bottom: 6px;
  padding-top: 10px;
}
.s7ecatalogviewer .s7linkdialog .s7dialogfooter .s7button {
  box-shadow: 1px 1px 1px #999999;
  color: #FFFFFF;
  font-size: 9pt;
  font-weight: bold;
  line-height: 34px;
  margin-right: 10px;
}
.s7ecatalogviewer .s7linkdialog .s7dialogcancelbutton {
  width: 64px;
  height: 34px;
}
.s7ecatalogviewer .s7linkdialog .s7dialogcancelbutton[state='up'] {
  background-color: #666666;
  color: #dddddd;
}
.s7ecatalogviewer .s7linkdialog .s7dialogcancelbutton[state='down'] {
  background-color: #555555;
  color: #ffffff;
}
.s7ecatalogviewer .s7linkdialog .s7dialogcancelbutton[state='over'] {
  background-color: #555555;
  color: #ffffff;
}
.s7ecatalogviewer .s7linkdialog .s7dialogcancelbutton[state='disabled'] {
  background-color: #b2b2b2;
  color: #dddddd;
}
.s7ecatalogviewer .s7linkdialog .s7dialogactionbutton {
  width: 82px;
  height: 34px;
}
.s7ecatalogviewer .s7linkdialog .s7dialogactionbutton[state='up'] {
  background-color: #333333;
  color: #dddddd;
}
.s7ecatalogviewer .s7linkdialog .s7dialogactionbutton[state='down'] {
  background-color: #222222;
  color: #cccccc;
}
.s7ecatalogviewer .s7linkdialog .s7dialogactionbutton[state='over'] {
  background-color: #222222;
  color: #cccccc;
}
.s7ecatalogviewer .s7linkdialog .s7dialogactionbutton[state='disabled'] {
  background-color: #b2b2b2;
  color: #dddddd;
}
```

The main dialog area (between the header and the footer) contains dialog content. In all cases, the component manages the width of this area—it is not possible to set it in CSS. Main dialog area is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkdialog .s7dialogviewarea
```

CSS properties of the dialog box viewing area

height	The height of the main dialog box area. It should be specified only when the dialog box works in desktop mode. It is not applicable when the dialog box is sized to occupy the entire browser window.
background-color	The background color of the main dialog box area.
margin	Outer margin.

Example – to set up a main dialog box area to be 300 pixels height, have a ten pixel margin, and use a white background:

```
.s7ecatalogviewer .s7linkdialog .s7dialogviewarea {
  background-color:#ffffff;
  margin:10px;
  height:300px;
}
```

All form content (like labels and input fields) resides inside a container controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkdialog .s7dialogbody
```

CSS properties of the dialog box body

padding	Inner padding.
---------	----------------

Example – to set up form content to have ten pixel padding:

```
.s7ecatalogviewer .s7linkdialog .s7dialogbody {
  padding: 10px;
}
```

All static labels in the dialog box form are controlled with

```
.s7ecatalogviewer .s7linkdialog .s7dialoglabel
```

This class is not suitable for controlling the label size or position because you can apply it to texts in various places of the form user interface.

CSS properties of the dialog box label.

font-weight	Label font weight.
font-size	Label font size.
font-family	Label font family.
color	Label text color.

Dialog box labels can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up all labels to be gray, bold with a nine pixel font:

```
.s7ecatalogviewer .s7linkdialog .s7dialoglabel {
  color: #666666;
  font-size: 9pt;
  font-weight: bold;
}
```


The size of the text copy displayed on top of the link is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkdialog .s7dialoginputwide
```

CSS properties of the dialog box input wide field

width	Text width.
padding	Inner padding.

Example – to set text copy to be 430 pixels wide and have a ten pixel padding in the bottom:

```
.s7ecatalogviewer .s7linkdialog .s7dialoginputwide {
  padding-bottom: 10px;
  width: 430px;
}
```

The share link is wrapped in a container and controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkdialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

border	Border around the share link container.
padding	Inner padding.

Example – to set a one pixel grey border around embed code text and have nine pixels of padding:

```
.s7ecatalogviewer .s7linkdialog .s7dialoginputcontainer {
  border: 1px solid #CCCCCC;
  padding: 9px;
}
```

The share link itself is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7linkdialog .s7dialoglink
```

CSS properties of the dialog box share link

width	Share link width.
-------	-------------------

Example – to set the share link to be 450 pixels wide:

```
.s7ecatalogviewer .s7linkdialog .s7dialoglink {
  width: 450px;
}
```

Main control bar

The main control bar is the rectangular area on desktop systems and tablets that contain all user interface controls (except Large Page buttons) available for the eCatalog viewer.

On mobile phones, it still keeps Thumbnails, Table of Contents, Download, Print, Favorites, Social share, Full Screen, and Close buttons. However, First Page and Last Page buttons, and Page Indicator are removed from the main control bar and added to the secondary control bar instead. By default, the main control bar is displayed in the top of the viewer area on desktop systems and mobile phones, and moved to the bottom of the viewer area on tablets. It always takes the entire available viewer width. It is possible to change its color, height, and vertical position in the CSS, relative to the viewer container.

The appearance of the main control bar is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7controlbar
```

CSS property	Description
top	Position from the top of the viewer.
bottom	Position from the bottom of the viewer.
height	The height of the main control bar.
background-color	The background color of the main control bar.

Example – to set up a gray main control bar that is 36 pixels tall and is positioned at the top of the viewer container.

```
.s7ecatalogviewer .s7controlbar {
  top: 0px;
  height: 36px;
  background-color: rgba(0, 0, 0, 0.5);
}
```

The main control bar supports an optional scroll feature. It is activated if the viewer width is too small and there is not enough space to fit all the buttons preset in the control bar. In this case, a two-state arrow button appears in the right-hand side of the control bar. Clicking or tapping on this button scrolls all the control bar elements to the left or to the right, depending on the scroll button state. The primary use case for this feature are mobile devices with small screens in portrait orientation.

The scroll feature is enabled for the main control bar, and is disabled for the secondary control bar. The feature is turned on and off using the following CSS class selector:

```
.s7ecatalogviewer .s7controlbar .s7innercontrolbarcontainer
```

CSS property	Description
position	When set to <code>static</code> the scroll feature is disabled. Set this property to <code>absolute</code> to enable the scroll feature.

The scroll button is added to a special container element that positions the button properly and lets you style the area around the button differently from the rest of the control bar background in case the height of the scroll button is smaller than the control bar height.

The appearance of this scroll button container is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7controlbar .s7scrollbuttoncontainer
```

CSS property	Description
width	Normally should be equal or larger than the width of the scroll button itself.
background-color	Container background color.

You can size and skin the scroll button itself by way of CSS.

The appearance of this button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7controlbar .s7scrolllefttrightbutton
```

CSS property	Description
width	Width of button.
height	Height of button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` and `selected` attribute selectors, which can be used to apply different skins to different button states. In particular, `state="selected"` corresponds to the initial scroll button state when it is possible to scroll control bar contents to the left; `state="default"` corresponds to the state when the content is scrolled all the way to the left and the scroll button suggests to return it to the initial state.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to enable the scroll feature in the main control bar for mobile phones, and set up a scroll button that is 64 x 64 pixels that displays a different image for each of the 4 different button states when selected or not selected:

```
.s7ecatalogviewer.s7size_small .s7controlbar .s7innercontrolbarcontainer {
  position: absolute;
}
.s7ecatalogviewer.s7size_small.s7touchinput .s7controlbar .s7scrollbuttoncontainer {
  width:64px;
  background-color: rgb(0, 0, 0);
}
.s7ecatalogviewer.s7size_small.s7touchinput .s7controlbar .s7scrolllefttrightbutton {
  width:64px;
  height:64px;
}
.s7ecatalogviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='true'][state='up'] {
  background-image:url(images/v2/ControlBarLeftButton_dark_up_touch.png);
}
.s7ecatalogviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='true'][state='over'] {
  background-image:url(images/v2/ControlBarLeftButton_dark_over_touch.png);
}
.s7ecatalogviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='true'][state='down'] {
  background-image:url(images/v2/ControlBarLeftButton_dark_down_touch.png);
}
.s7ecatalogviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='true'][state='disabled'] {
  background-image:url(images/v2/ControlBarLeftButton_dark_disabled_touch.png);
}
.s7ecatalogviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='false'][state='up'] {
  background-image:url(images/v2/ControlBarRightButton_dark_up_touch.png);
}
```

```
.s7ecatalogviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='false'][state='over'] {
  background-image:url(images/v2/ControlBarRightButton_dark_over_touch.png);
}
.s7ecatalogviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='false'][state='down'] {
  background-image:url(images/v2/ControlBarRightButton_dark_down_touch.png);
}
.s7ecatalogviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='false'][state='disabled'] {
  background-image:url(images/v2/ControlBarRightButton_dark_disabled_touch.png);
}
```

Main viewer area

The main view area is the area occupied by the catalog image. It usually sets to fit the available device screen when no size is specified.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7ecatalogviewer
```

CSS property	Description
width	The width of the viewer.
height	The height of the viewer.
background-color	Background color in hexadecimal format.

Example – to set up a viewer with a white background (#FFFFFF) and make its size 512 x 288 pixels.

```
.s7ecatalogviewer {
  background-color: #FFFFFF;
  width: 512px;
  height: 288px;
}
```

Next page button

Clicking or tapping on this button brings the user to the next page in the catalog. This button appears in the main control bar. This button is not displayed on mobile phones to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7toolbarrightbutton .s7panrightbutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.

CSS property	Description
left	Position from the left border of the main control bar, including padding.
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a next page button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 250 pixels from the right edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7toolbarrightbutton .s7panrightbutton {
bottom:4px;
right:250px;
width:32px;
height:32px;
}
.s7ecatalogviewer .s7toolbarrightbutton .s7panrightbutton [state='up'] {
background-image:url(images/v2/ToolBarRightButton_dark_up.png);
}
.s7ecatalogviewer .s7toolbarrightbutton .s7panrightbutton [state='over'] {
background-image:url(images/v2/ToolBarRightButton_dark_over.png);
}
.s7ecatalogviewer .s7toolbarrightbutton .s7panrightbutton [state='down'] {
background-image:url(images/v2/ToolBarRightButton_dark_down.png);
}
.s7ecatalogviewer .s7toolbarrightbutton .s7panrightbutton [state='disabled'] {
background-image:url(images/v2/ToolBarRightButton_dark_disabled.png);
}
```

Page indicator

Page indicator displays current page index and total page count. It appears in main control bar on desktop systems and tablet, on mobile phones it is added to secondary control bar. Page indicator can be sized, skinned, and positioned by CSS.

The appearance page indicator is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7pageindicator
```

CSS property	Description
top	Position from the top border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
right	Position from the right border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
left	Position from the left border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
bottom	Position from the bottom border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
width	Width of the page indicator.
height	Height of the page indicator.
color	Font color.
font-family	Font name.
font-size	Font size.

Example – to set up a page indicator that is 56 x 28 pixels, horizontally centered and positioned 4 pixels from the bottom of the main control bar, and use a 14 pixel Helvetica font.

```
.s7ecatalogviewer .s7pageindicator {
  position:absolute;
  bottom: 4px;
  margin-left: -28px;
  left: 50%;
  width:56px;
  height:28px;
  font-family: Helvetica;
  font-size:14px;
}
```

Page view

Main view consists of the catalog image. It can be swiped to get to another page or zoomed.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7pageview
```

CSS property	Description
background-color	Background color of the main view in hexadecimal format.

CSS property	Description
cursor	Cursor that is displayed over the main view.

Example – to make the main view transparent.

```
.s7ecatalogviewer .s7pageview {
  background-color: transparent;
}
```

On desktop systems the component supports the `cursor` attribute selector which can be applied to `.s7pageview` class and controls the type of the cursor based on component state and user action. The following `cursor` values are supported:

Value	Description
default	Displayed when the image is not zoomable because of a small image resolution, component settings, or both.
zoomin	Displayed when the image can be zoomed in.
reset	Displayed when the image is at maximum zoom level and can be reset to initial state.
drag	Displayed when user pans the image which is in zoomed in state.
slide	Displayed when the user performs an image swap by doing horizontal swipe or flick.

The page divider that visually separates the left and right pages of the catalog spread is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7pageview .s7pagedivider
```

CSS property	Description
width	The width of page divider. Set to 0 px to hide the divider completely.
background-image	The image that you want to use as the page divider.

Example – to have 40 pixels wide page divider with semi-transparent image.

```
.s7ecatalogviewer .s7pageview .s7pagedivider {
  width: 40px;
  background-image: url(images/sdk/divider.png);
}
```



Note: When the `frametransition` modifier is set to `turn` or `auto` (on desktop systems), the appearance of the page divider is controlled with the `pageturnstyle` modifier and the `.s7pagedivider` CSS class is ignored.

It is possible to configure the display of the custom mouse cursors over the main viewer area. This is controlled with the additional attribute selectors applied to `.s7ecatalogviewer .s7pageview` CSS class:

CSS property	Description
default	Normally an arrow, displays for non-zoomable image.
zoomin	Shows when an image can be zoomed in.
reset	Shows when an image is at maximum zoom and can be reset.
drag	Shows when user performs drag operation on zoomed in image
slide	Shows when user performs image swap using slide gesture

Example – have different mouse cursors for each type of component state.

```
.s7ecatalogviewer .s7pageview[cursortype="default"] {
  cursor:auto;
}
.s7ecatalogviewer .s7pageview[cursortype="zoomin"] {
  cursor:url(images/zoomin_cursor.cur), auto;
}
.s7ecatalogviewer .s7pageview[cursortype="reset"] {
  cursor:url(images/zoomout_cursor.cur), auto;
}
.s7ecatalogviewer .s7pageview [cursortype="slide"] {
  cursor:url(images/slide_cursor.cur), auto;
}
.s7ecatalogviewer .s7pageview[cursortype="drag"] {
  cursor:url(images/drag_cursor.cur), auto;
}
```

Previous page button

Clicking or tapping on this button brings the user to the previous page in the catalog. This button appears in the main control bar. This button is not displayed on mobile phones to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7toolbarleftbutton .s7panleftbutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.
left	Position from the left border of the main control bar, including padding.

CSS property	Description
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a previous page button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 250 pixels from the right edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7toolbarleftbutton .s7panleftbutton {
bottom:4px;
left:250px;
width:32px;
height:32px;
}
.s7ecatalogviewer .s7toolbarleftbutton .s7panleftbutton [state='up'] {
background-image:url(images/v2/ToolBarLeftButton_dark_up.png);
}
.s7ecatalogviewer .s7toolbarleftbutton .s7panleftbutton [state='over'] {
background-image:url(images/v2/ToolBarLeftButton_dark_over.png);
}
.s7ecatalogviewer .s7toolbarleftbutton .s7panleftbutton [state='down'] {
background-image:url(images/v2/ToolBarLeftButton_dark_down.png);
}
.s7ecatalogviewer .s7toolbarleftbutton .s7panleftbutton [state='disabled'] {
background-image:url(images/v2/ToolBarLeftButton_dark_disabled.png);
}
```

Print

Print tool consists of a button added to the control bar and the modal dialog box that displays when the tool is activated.

The appearance of the print button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7print
```

CSS properties of the print button

margin-top	The offset from the top of the control bar.
margin-left	The distance to the next button on the left, or the left side of the control bar if this is the first button in a row.

width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – To set up a print button that is 28 x 28 pixels, and displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7print {
margin-top: 4px;
margin-left: 10px;
width: 28px;
height: 28px;
}
.s7ecatalogsearchviewer .s7print[state='up'] {
background-image: url(images/v2/Print_dark_up.png);
}
.s7ecatalogsearchviewer .s7print[state='over'] {
background-image: url(images/v2/Print_dark_over.png);
}
.s7ecatalogsearchviewer .s7print[state='down'] {
background-image: url(images/v2/Print_dark_down.png);
}
.s7ecatalogsearchviewer .s7print[state='disabled'] {
background-image: url(images/v2/Print_dark_disabled.png);
}
```

The background overlay which covers the web page when the dialog box is active is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7backoverlay
```

CSS properties of the back overlay

opacity	Background overlay opacity.
background-color	Background overlay color.

Example – to set up background overlay to be gray with 70% opacity:

```
.s7ecatalogsearchviewer .s7printdialog .s7backoverlay {
opacity: 0.7;
background-color: #222222;
}
```

By default the modal dialog is displayed centered in the screen on desktop systems. The positioning and sizing of the dialog box is managed by the component. The dialog is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7kprintdialog .s7dialog
```

CSS properties of the dialog box

border-radius	Dialog box border radius.
background-color	Dialog box background color;

Example – to set up a dialog box to have a grey background:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialog {
background-color: #dddddd;
}
```

The dialog box header consists of an icon, a title text, and a close button. The header container is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogheader
```

CSS properties of the dialog box header

padding	Inner padding for header content.
---------	-----------------------------------

The icon and the title text are wrapped into an additional container controlled with the following:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogheader .s7dialogline
```

CSS properties of the dialog line

padding	Inner padding for the header icon and title.
---------	--

Header icon is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogheadericon
```

CSS properties of the dialog box header icon

width	Icon width.
height	Icon height.
background-image	Icon image.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .

Header title is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogheadertext
```

CSS properties of the dialog box header text

font-weight	Font weight.
-------------	--------------

font-size	Font height.
font-family	Font family.
padding	Internal text padding.

Close button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7closebutton
```

CSS properties of the close button

top	Vertical button position relative to header container.
right	Horizontal button position relative to header container.
width	Button width.
height	Button height.
padding	Inner padding of button.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The Close button tool tip and the dialog box title can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up dialog header with padding, 22 x 22 pixels icon, bold 16 point title, and a 28 x 28 pixel Close button positioned two pixels from the top and two pixels from the right of dialog box container:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogheader {
  padding: 10px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogheader .s7dialogline {
  padding: 10px 10px 2px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogheadericon {
  background-image: url("images/sdk/dlgprint_cap.png");
  height: 22px;
  width: 22px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogheadertext {
  font-size: 16pt;
  font-weight: bold;
  padding-left: 16px;
}
.s7ecatalogsearchviewer .s7printdialog .s7closebutton {
  top: 2px;
  right: 2px;
```

```
padding:8px;
width:28px;
height:28px;
}
.s7ecatalogsearchviewer .s7printdialog .s7closebutton[state='up'] {
background-image:url(images/sdk/close_up.png);
}
.s7ecatalogsearchviewer .s7printdialog .s7closebutton[state='over'] {
background-image:url(images/sdk/close_over.png);
}
.s7ecatalogsearchviewer .s7printdialog .s7closebutton[state='down'] {
background-image:url(images/sdk/close_down.png);
}
.s7ecatalogsearchviewer .s7printdialog .s7closebutton[state='disabled'] {
background-image:url(images/sdk/close_disabled.png);
}
```

Dialog box footer consists of Cancel and Send to Print buttons. The footer container is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogfooter
```

CSS properties of the dialog box footer

border	Border that you can use to visually separate the footer from the rest of the dialog box.
--------	--

The footer has an inner container that keeps both buttons. It is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogbuttoncontainer
```

CSS properties of the dialog box button container


padding	Inner padding between the footer and the buttons.
---------	---

Cancel button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogcancelbutton
```

CSS properties of the dialog box cancel button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.

 **Note:** This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

Send to Print button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogactionbutton
```

CSS properties of the dialog box action button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

In addition, both buttons share the same common CSS class which can contain CSS settings that are the same for other dialog box buttons:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogfooter .s7button
```

CSS properties of the button

font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.
line-height	Text height inside the button. Affects vertical alignment.
box-shadow	Drop shadow.
margin-right	Right button margin.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box footer with 64 x 34 Cancel button and a 96 x 34 Send to Print button, with the text color and background color different for each button state:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogfooter {
  border-top: 1px solid #909090;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogbuttoncontainer {
  padding-bottom: 6px;
  padding-top: 10px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogfooter .s7button {
  box-shadow: 1px 1px 1px #999999;
  color: #FFFFFF;
  font-size: 9pt;
  font-weight: bold;
  line-height: 34px;
  margin-right: 10px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogcancelbutton {
  width: 64px;
  height: 34px;
}
```

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogcancelbutton[state='up'] {
  background-color:#666666;
  color:#dddddd;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogcancelbutton[state='down'] {
  background-color:#555555;
  color:#ffffff;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogcancelbutton[state='over'] {
  background-color:#555555;
  color:#ffffff;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogcancelbutton[state='disabled'] {
  background-color:#b2b2b2;
  color:#dddddd;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogactionbutton {
  width:96px;
  height:34px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogactionbutton[state='up'] {
  background-color:#333333;
  color:#dddddd;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogactionbutton[state='down'] {
  background-color:#222222;
  color:#cccccc;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogactionbutton[state='over'] {
  background-color:#222222;
  color:#cccccc;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogactionbutton[state='disabled'] {
  background-color:#b2b2b2;
  color:#dddddd;
}
```

The main dialog area (between the header and the footer) contains dialog content. In all cases, the component manages the width of this area, it is not possible to set it in CSS. The main dialog area is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogviewarea
```

CSS properties of the dialog box viewing area

height	The height of the main dialog box area.
background-color	The background color of the main dialog box area.
margin	Outer margin.

Example – to set up a main dialog area to have an automatically calculated height, have a ten pixel margin, and use a white background:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogviewarea {
  background-color:#ffffff;
  margin:10px;
  height:auto;
}
```

All form content (like labels and input fields) resides inside a container controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogbody
```

CSS properties of the dialog box body

padding	Inner padding.
---------	----------------

Example – to set up form content to have ten pixel padding:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogbody {
    padding: 10px;
}
```

Dialog box form is filled on line-by-line basis, where each line carries a part of the form content (like a label and a text input field). Single form line is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogbody .s7dialogline
```

CSS properties of the dialog box line

padding	Inner line padding.
---------	---------------------

Example – to set up a dialog box form to have ten pixel padding for each line:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogbody .s7dialogline {
    padding: 10px;
}
```

The size of the block of dialog content is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialoginputwide
```

CSS properties of the dialog input width

width	Block width.
padding	Inner line padding.

Example – to set a content block to be 430 pixels wide and have 10 pixels padding in the bottom:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialoginputwide {
    padding-bottom: 10px;
    width: 430px;
}
```

All static labels in the dialog box form are controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialoglabel
```

This class is not suitable for controlling the label size or position because you can apply it to texts in various places in the form user interface.

CSS properties of the dialog box label.

font-weight	Label font weight.
font-size	Label font size.
font-family	Label font family.
color	Label text color.

Dialog box labels can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up all labels to be gray, bold, with a nine pixel font:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialoglabel {
  color: #666666;
  font-size: 9pt;
  font-weight: bold;
}
```

Input controls are wrapped into the container and controlled with the following CSS class selector::

```
.s7ecatalogsearchviewer .s7printdialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

padding-left	Inner padding.
--------------	----------------

Example – to set a 30 pixel padding from the left edge of the dialog box.

```
.s7ecatalogsearchviewer .s7printdialog .s7dialoginputcontainer {
  padding-left: 30px;
}
```

Radio buttons and their caption text are controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogoption
```

CSS properties of the dialog box option

width	The total width of the radio button with a caption.
color	Caption text color.

The spacing between the radio button and its caption is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogoptioninput
```

CSS properties of the dialog box option input

margin-right	Spacing between the radio button and its caption.
--------------	---

Numeric pickers for print range selection are controlled with the following CSS class selector

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogrange
```

CSS properties of the dialog box print range

width	Width of the numeric picker.
margin	Spacing around the numeric picker.

Example – to set up all radio buttons to be 150 pixels wide with black text, ten pixel spacing, and 42 pixel wide numeric pickers:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogoption {
  color: #000000;
  width: 150px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogoption input {
```

```

    margin-right: 10px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogrange {
    margin-left: 10px;
    margin-right: 10px;
    width: 42px;
}

```

The horizontal divider between the page range selection and the print layout sections is controlled with the following CSS class selector:

```

.s7ecatalogsearchviewer
.s7printdialog .s7horizontaldivider

```

CSS properties of the horizontal divider

border	Border around divider.
padding	Inner padding.
width	Divider width.
margin	Outer margin

Example – to set up a 430 pixel wide grey divider with a 10 pixel vertical padding on both sides, and a ten pixel margin at the top:

```

.s7ecatalogsearchviewer .s7printdialog .s7horizontaldivider {
    border-top: 1px solid #aaaaaa;
    margin-top: 10px;
    padding-bottom: 10px;
    padding-top: 10px;
    width: 430px;
}

```

Remove Favorite button

The position of the Remove Favorite button is fully managed by the Favorites menu.

The appearance of the Remove Favorite button is controlled with the following CSS class selector:

```

.s7ecatalogviewer .s7removefavoritebutton

```

CSS properties of the Remove Favorite button

background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
width	Width of the button.
height	Height of the button.



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the state when a user can add a new Favorite icon by clicking or tapping. *selected='false'* corresponds to the normal operation mode when a user can zoom, pan, and swap pages.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Remove Favorite button that is 28 x 28 pixels, and displays a different image for each of the four different button states when selected or not selected.

```
.s7ecatalogviewer .s7removefavoritebutton {
  width:28px;
  height:28px;
}
.s7ecatalogviewer .s7removefavoritebutton[selected='false'][state='up'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_up.png);
}
.s7ecatalogviewer .s7removefavoritebutton[selected='false'][state='over'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_over.png);
}
.s7ecatalogviewer .s7removefavoritebutton[selected='false'][state='down'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_down.png);
}
.s7ecatalogviewer .s7removefavoritebutton[selected='false'][state='disabled'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_disabled.png);
}
.s7ecatalogviewer .s7removefavoritebutton[selected='true'][state='up'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_over.png);
}
.s7ecatalogviewer .s7removefavoritebutton[selected='true'][state='over'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_over.png);
}
.s7ecatalogviewer .s7removefavoritebutton[selected='true'][state='down'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_over.png);
}
.s7ecatalogviewer .s7removefavoritebutton[selected='true'][state='disabled'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_disabled.png);
}
```

Secondary control bar

The secondary control bar is the rectangular area that contains First and Last Page buttons and a Page Indicator when made available in CSS.

By default, it is displayed on mobile phones only and is located in the bottom of the viewer. It always takes the whole available viewer width. It is possible to change its color, height and vertical position by CSS, relative to the viewer container.

The appearance of the secondary control bar is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7secondarycontrols .s7controlbar
```

CSS property	Description
top	Position from the top of the viewer.
bottom	Position from the bottom of the viewer.
height	The height of the main control bar.

CSS property	Description
background-color	The background color of the secondary control bar.

Example – to set up a gray secondary control bar that is 72 pixels tall and is positioned at the bottom of the viewer container.

```
.s7ecatalogviewer .s7secondarycontrols .s7controlbar {
  bottom: 0px;
  height: 72px;
}
```

Social share

The social share tool appears in the top left corner by default. It consists of a button and a panel which expands when user clicks or taps on a button and contains individual sharing tools.

The position and size of the social share tool in the viewer user interface is controlled with the following:

```
.s7ecatalogviewer .s7socialshare
```

CSS properties of the social share tool

margin-top	The offset from the top of the control bar.
margin-left	The distance to the next button on the left, or the left side of the control bar if this is the first button in a row.
width	The width of the social sharing tool.
height	The height of the social sharing tool.

Example – set up a social sharing tool that is positioned four pixels from the top and five pixels from the right of viewer container and is sized to 28 x 28 pixels.

```
.s7ecatalogviewer .s7socialshare {
  margin-top: 4px;
  margin-left: 10px; width:28px;
  height:28px;
}
```

The appearance of the social share tool button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7socialshare .s7socialbutton
```

CSS properties of the social button

background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – set up a social sharing tool button that displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7socialshare .s7socialbutton[state='up'] {
background-image:url(images/v2/SocialShare_video_dark_up.png);
}
.s7ecatalogviewer .s7socialshare .s7socialbutton[state='over'] {
background-image:url(images/v2/SocialShare_dark_over.png);
}
.s7ecatalogviewer .s7socialshare .s7socialbutton[state='down'] {
background-image:url(images/v2/SocialShare_dark_down.png);
}
.s7ecatalogviewer .s7socialshare .s7socialbutton[state='disabled'] {
background-image:url(images/v2/SocialShare_dark_disabled.png);
}
```

The appearance of the panel which contains the individual social sharing icons is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7socialshare .s7socialsharepanel
```

CSS properties of the social share panel

background-color	The background color of the panel.
------------------	------------------------------------

Example – set up a panel to have transparent color:

```
.s7ecatalogviewer .s7socialshare .s7socialsharepanel {
background-color: transparent;
}
```

Table of contents

Table of contents is a button located in the main control bar. When activated, a drop-down panel appears with a list of page indexes and labels.

Based on the configuration, the list can contain all pages that are present in the catalog or only those pages that have explicit labels defined. On desktop systems, if the list is longer than the available screen real estate, a scroll bar is displayed on the right.


The position and size of the table of contents button in the viewer user interface is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7tableofcontents
```

CSS properties of the table of contents

margin-top	The offset from the top of the control bar.
margin-left	The distance to the next button on the left, or the left side of the control bar if this is the first button in a row.
width	The width of the table of contents button.
height	The height of the table of contents button.
background-image	The image that is displayed for a given button state.

background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
---------------------	--

 **Note:** This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – set up a table of contents button that is positioned 4 pixels from the bottom and 43 pixels from the left of the main control bar; size is 28 x 28 pixels and a different image is displayed for each of the four different button states:


```
.s7ecatalogviewer .s7tableofcontents {
margin-top: 4px;
margin-left: 10px; width: 28px;
height: 28px;
}
.s7ecatalogviewer .s7tableofcontents[state='up'] {
background-image:url(images/v2/TableOfContents_dark_up.png);
}
.s7ecatalogviewer .s7tableofcontents[state='over'] {
background-image:url(images/v2/TableOfContents_dark_over.png);
}
.s7ecatalogviewer .s7tableofcontents[state='down'] {
background-image:url(images/v2/TableOfContents_dark_down.png);
}
.s7ecatalogviewer .s7tableofcontents[state='disabled'] {
background-image:url(images/v2/TableOfContents_dark_disabled.png);
}
```

The appearance of the drop-down panel is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7tableofcontents .s7panel
```

CSS properties of the drop-down panel

background-color	Background color of drop-down panel.
margin	Internal offset between the panel boundaries and the content.
box-shadow	Drop-shadow around the panel.

 **Note:** It is not possible to control the size or the position of the drop-down panel from CSS; the component manages its layout programmatically.

Example – set up a drop-down panel that has a semi-transparent black background, a 5 pixel margin around the content, and a drop shadow:

```
.s7ecatalogviewer .s7tableofcontents .s7panel {
background-color: rgba(0, 0, 0, 0.5);
margin: 5px;
box-shadow: 2px 2px 3px #c0c0c0;
}
```

The individual item look and feel is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7tableofcontents .s7panel .s7item
```

CSS properties of the item

font-family	Font name.
font-size	Font size.
height	Item's height.
padding	Internal padding.



Note: Drop-down list item supports the `state` attribute selector, which can be used to apply different skins to hover and selected item states.

Example – set up a drop-down item with a Helvetica 14 pixel font and 19 pixel high. An item has a dark gray background on hover and a light gray background when selected:

```
.s7ecatalogviewer .s7tableofcontents .s7panel .s7item {
font-family: Helvetica,sans-serif;
font-size: 14px;
height: 19px;
}
.s7ecatalogviewer .s7tableofcontents .s7panel .s7item[state="over"] {
background-color: rgb(102, 102, 102);
}
.s7ecatalogviewer .s7tableofcontents .s7panel .s7item[state="selected"] {
background-color: rgb(178, 178, 178);
}
```

An element that shows the page index is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7tableofcontents .s7panel .s7index
```

CSS properties of the page index

min-width	Minimum element width.
max-width	Maximum element width.
padding-right	Distance between the page index and the page label.



Note: It is possible to hide the page index entirely by setting `display:none` for the `s7index` CSS class.

Example 1 – set up a page index with a minimum width of 40 pixels, a maximum width of 70 pixels, and a 5 pixel margin on the right-hand side:

```
.s7ecatalogviewer .s7tableofcontents .s7panel .s7index {
max-width: 70px;
min-width: 40px;
padding-right: 5px;
}
```

Example 2 – hide page index:

```
.s7ecatalogviewer .s7tableofcontents .s7panel .s7index {
display: none;
}
```

The page label is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7tableofcontents .s7panel .s7label
```

CSS properties of the page label

min-width	Minimum element width.
max-width	Maximum element width.

Example – set up a page index with a minimum width of 40 pixels and a maximum width of 240 pixels:

```
.s7ecatalogviewer .s7tableofcontents .s7panel .s7label {
  min-width: 40px;
  max-width: 240px;
}
```

In case there are more items than can fit vertically within the drop-down panel and the system is a desktop, the component renders a vertical scroll bar on the right side of the panel. The appearance of the scroll bar area is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7tableofcontents .s7scrollbar
```

CSS properties of the scrollbar

width	The scroll bar width.
top	The vertical scroll bar offset from the top of the panel area.
bottom	The vertical scroll bar offset from the bottom of the panel area.
right	The horizontal scroll bar offset from the right edge of the panel area.

Example – set up a scroll bar that is 28 pixels wide and does not have a margin for the top, right, or bottom area of the panel:

```
.s7ecatalogviewer .s7tableofcontents .s7scrollbar {
  top: 0px;
  bottom: 0px;
  right: 0px;
  width: 28px;
}
```

Scroll bar track is the area between the top and bottom scroll buttons. The component automatically sets the position and height of the track. The track is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrolltrack
```

CSS properties of the scroll track

width	The track width.
background-color	The track background color.

Example – set up a scroll bar track that is 28 pixels wide and has a semi-transparent gray background:

```
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrolltrack {
  width:28px;
  background-color:rgba(102, 102, 102, 0.5);
}
```

The scroll bar thumb moves vertically within the scroll track area. Its vertical position is controlled by the component logic. However, thumb height does not dynamically change depending on the amount of content. You can configure the thumb height and other aspects with the following CSS class selector:

```
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrollthumb
```

CSS properties of the scrollbar thumb

width	The thumb width.
height	The thumb height.
padding-top	The vertical padding between the top of the track.
padding-bottom	The vertical padding between the bottom of the track.
background-image	The image that is displayed for a given thumb state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note:

Thumb supports the *state* attribute selector, which can be used to apply different skins to the *up*, *down*, *over*, and *disabled* thumb states.

Example – set up a scroll bar thumb that is 28 x 45 pixels, has 10 pixel margins on the top and bottom, and has different artwork for each state:

```
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrollthumb {
  width:28px;
  background-repeat:no-repeat;
  background-position:center;
  height:45px;
  padding-top:10px;
  padding-bottom:10px;
}
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrollthumb[state='up'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_up.png);
}
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrollthumb[state='down'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_down.png);
}
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrollthumb[state='over'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_over.png);
}
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrollthumb[state='disabled'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_up.png);
}
```

The appearance of the top and bottom scroll buttons is controlled with the following CSS class selectors:

```
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrollupbutton
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrolldownbutton
```

It is not possible to position the scroll buttons using CSS `top`, `left`, `bottom`, and `right` properties; instead, the viewer logic positions them automatically.

CSS properties of the scroll up and scroll down button

width	The button width.
height	The button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note:

Button supports the `state` attribute selector, which can be used to apply different skins to the up, down, over, and disabled button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – set up scroll buttons that are 28 x 32 pixels and have different artwork for each state:

```
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrollupbutton {
  width:28px;
  height:32px;
}
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrollupbutton[state='up'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_up.png);
}
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrollupbutton[state='over'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_over.png);
}
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrollupbutton[state='down'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_down.png);
}
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrollupbutton[state='disabled'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_up.png);
}
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrolldownbutton {
  width:28px;
  height:32px;
}
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrolldownbutton[state='up'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_up.png);
}
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrolldownbutton[state='over'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_over.png);
}
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrolldownbutton[state='down'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_down.png);
}
.s7ecatalogviewer .s7tableofcontents .s7scrollbar .s7scrolldownbutton[state='disabled'] {
```

```
background-image:url(images/v2/ThumbnailScrollDownButton_dark_up.png);
}
```

Thumbnails

Thumbnails consist of a grid of thumbnail images with an optional scroll bar on the right side to allow vertical scrolling.

Thumbnails are toggled by clicking the thumbnail button in the main control bar. When thumbnails are active, they display in modal mode overlaid on top of the viewer user interface. The viewer logic automatically resizes the thumbnails container to the entire viewer area.

The appearance of the thumbnails container is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7thumbnailgridview
```

CSS property	Description
top	The vertical offset of the thumbnails container from the top of the viewer.
margin-top	The top margin.
margin-left	The left margin.
margin-right	The right margin.
margin-bottom	The bottom margin.
background-color	The background color of the thumbnails area.

Example – to set up thumbnails to have 32 pixels offset from the top, 5 pixels margins on the left and right, and 8 pixels margin on the bottom, with 0xDDDDDD background.

```
.s7ecatalogviewer .s7thumbnailgridview {
  top: 32px;
  margin-left: 5px;
  margin-right: 5px;
  margin-bottom: 8px;
  background-color: rgb(221, 221, 221);
}
```

The spacing between thumbnails is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7thumbnailgridview .s7thumbcell
```

CSS property	Description
margin	The size of the horizontal and vertical margin around each thumbnail. Actual horizontal thumbnail spacing is equal to the sum of the left and right margin that is set for <code>.s7thumbcell</code> . Vertical thumbnail spacing equals to the sum of the top and bottom margin.

Example – to set a 10 pixel space both vertically and horizontally.

```
.s7ecatalogviewer .s7thumbnailgridview .s7thumbcell {
  margin: 5px;
}
```

The appearance of individual thumbnail is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7thumbnailgridview .s7thumb
```

CSS property	Description
width	The width of the thumbnail.
height	The height of the thumbnail.
border	The border of the thumbnail.
background-color	The background color of the thumbnail.

On touch devices, when rotated to portrait mode, the viewer may size thumbnails to half of what is configured in case it decides to split the catalog spread into individual pages.



Note: Thumbnail supports the state attribute selector, which can be used to apply different skins to different thumbnail states. In particular, `state="selected"` corresponds to the thumbnail for the image that is currently displayed in the main view, `state="default"` corresponds to the rest of thumbnails, and `state="over"` is used on mouse hover.

Example – to set up thumbnails that are 120 x 85 pixels, have a white background, a light gray standard border, and a dark grey selected border.

```
.s7ecatalogviewer .s7thumbnailgridview .s7thumb {
  width:120px;
  height:85px;
  background-color: rgb(255, 255, 255);
  border: solid 1px #999999;
}
.s7ecatalogviewer .s7thumbnailgridview .s7thumb[state="selected"]{
  border: solid 2px #666666;
}
```

The appearance of the thumbnail label is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7thumbnailgridview .s7label
```

CSS property	Description
font-family	Font name.
font-size	Font size.

Example – to set up labels to use 14 pixels Helvetica font.

```
.s7ecatalogviewer .s7thumbnailgridview .s7label {
  font-family: Helvetica,sans-serif;
```

```
font-size: 12px;
}
```

In case there are more thumbnails than can fit vertically into the view, thumbnails renders the vertical scroll bar on the right side. The appearance of scroll bar area is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar
```

CSS property	Description
width	The width of the scroll bar.
top	The vertical scroll bar offset from the top of the thumbnails area.
bottom	The vertical scroll bar offset from the bottom of the thumbnails area.
right	The horizontal scroll bar offset from the right edge of the thumbnails area.

Example – to set up a scroll bar that is 28 pixels wide and has an 8 pixel margin from the top, right, and bottom of the thumbnails area.

```
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar {
  top:8px;
  bottom:8px;
  right:8px;
  width:28px;
}
```

The scroll bar track is the area between the top and bottom scroll buttons. The component automatically sets the position and height of the track. The track is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrolltrack
```

CSS property	Description
width	The width of the scroll bar track.
background-color	The background color of the scroll bar track.

Example – to set up a scroll bar track that is 28 pixels wide and has a semi-transparent gray background.

```
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrolltrack {
  width:28px;
  background-color:rgba(102, 102, 102, 0.5);
}
```

The scroll bar thumb moves vertically within the scroll track area. Its vertical position is fully controlled by the component logic, however, thumb height does not dynamically change depending on the amount of content. The thumb height and other aspects are controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrollthumb
```

CSS property	Description
width	The width of the scroll bar thumb.
height	The height of the scroll bar thumbnail.
padding-top	The vertical padding between the top of the scroll bar track.
padding-bottom	The vertical padding between the bottom of the scroll bar track.
background-image	The image that is displayed for a given thumb state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: Thumb supports the *state* attribute selector, which can be used to apply different skins to the thumb states *up*, *down*, *over*, and *disabled*.

Example – to set up a scroll bar thumb that is 28 x 45 pixels, has 10 pixel margins on the top and bottom, and have different artwork for each state.

```
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrollthumb {
  width:28px;
  background-repeat:no-repeat;
  background-position:center;
  height:45px;
  padding-top:10px;
  padding-bottom:10px;
}
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrollthumb[state='up'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_up.png);
}
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrollthumb[state='down'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_down.png);
}
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrollthumb[state='over'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_over.png);
}
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrollthumb[state='disabled'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_up.png);
}
```

The appearance of the top and bottom scroll buttons is controlled with following CSS class selectors:

```
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrollupbutton
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrolldownbutton
```

It is not possible to position the scroll buttons using CSS *top*, *left*, *bottom*, and *right* properties. Instead, the viewer logic positions them automatically.

CSS property	Description
width	The width of the button.
height	The height of the button.
background-image	The image that is displayed for a given thumb state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: These buttons support the *state* attribute selector, which can be used to apply different skins to the different button states *up*, *down*, *over*, and *disabled*.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up scroll buttons which are 28 x 32 pixels and have different artwork for each state.

```
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrollupbutton {
  width:28px;
  height:32px;
}
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrollupbutton[state='up'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_up.png);
}
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrollupbutton[state='over'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_over.png);
}
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrollupbutton[state='down'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_down.png);
}
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrollupbutton[state='disabled'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_up.png);
}
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrolldownbutton {
  width:28px;
  height:32px;
}
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrolldownbutton[state='up'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_up.png);
}
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrolldownbutton[state='over'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_over.png);
}
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrolldownbutton[state='down'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_down.png);
}
.s7ecatalogviewer .s7thumbnailgridview .s7scrollbar .s7scrolldownbutton[state='disabled'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_up.png);
}
```

Thumbnails button

Clicking or tapping this button resets toggles the viewer between the main view and the thumbnails. This button appears in the main control bar. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7thumbnailpagebutton
```

CSS property	Description
margin-top	The offset from the top of the control bar.
margin-left	The distance to the next button on the left, or the left side of the control bar if this is the first button in a row.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the viewer state when thumbnail mode is active state and *selected='false'* corresponds to the default state with main view.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up thumbnails button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 5 pixels from the left edge of the main control bar, and displays a different image for each of the four different button states when selected or not selected.

```
.s7ecatalogviewer .s7thumbnailpagebutton{
margin-top: 4px;
margin-left: 5px; width:28px;
height:28px;
}
.s7ecatalogviewer .s7thumbnailpagebutton [selected='false'][state='up'] {
background-image:url(images/v2/ThumbnailPageButton_dark_up.png);
}
.s7ecatalogviewer .s7thumbnailpagebutton [selected='false'][state='over'] {
background-image:url(images/v2/ThumbnailPageButton_dark_over.png);
}
.s7ecatalogviewer .s7thumbnailpagebutton [selected='false'][state='down'] {
background-image:url(images/v2/ThumbnailPageButton_dark_down.png);
}
.s7ecatalogviewer .s7thumbnailpagebutton [selected='false'][state='disabled'] {
background-image:url(images/v2/ThumbnailPageButton_dark_disabled.png);
}
.s7ecatalogviewer .s7thumbnailpagebutton [selected='true'][state='up'] {
background-image:url(images/v2/ThumbnailPageButton_dark_over.png);
}
.s7ecatalogviewer .s7thumbnailpagebutton [selected='true'][state='over'] {
background-image:url(images/v2/ThumbnailPageButton_dark_over.png);
}
.s7ecatalogviewer .s7thumbnailpagebutton [selected='true'][state='down'] {
background-image:url(images/v2/ThumbnailPageButton_dark_over.png);
}
.s7ecatalogviewer .s7thumbnailpagebutton [selected='true'][state='disabled'] {
```



```
background-image:url(images/v2/ThumbnailPageButton_dark_disabled.png);
}
```

Tooltips

On desktop systems some user interface elements like buttons have tooltips that are displayed on mouse hover.

CSS properties of the main viewer area

The appearance of tooltips is controlled with the following CSS class selector:

```
.s7tooltip
```

CSS property	Description
border-radius	Background border radius.
border-color	Background border color.
background-color	Background color.
color	Text color.
font-family	Text font name.
font-size	Text font size.



Note: In case tooltip styles are customized from within the embedding web page, all properties have to contain !IMPORTANT rule. This is not necessary if tooltips are customized in the viewer's CSS file.

Example – to set up tooltips that have a grey border with 3px corner radius, black background and white text written with Arial, 11 pixels size:

```
.s7tooltip {
border-radius: 3px 3px 3px 3px;
border-color: #999999;
background-color: #000000;
color: #FFFFFF;
font-family: Arial, Helvetica, sans-serif;
font-size: 11px;
}
```

Twitter share

Twitter share tool consists of a button added to the Social share panel. When the button is clicked the user is redirected to a sharing dialog box that is provided by a social service. The position of the button is fully managed by the Social share tool.

The appearance of the Twitter share button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7twittershare
```

CSS properties of the Twitter share tool

width	Button width.
-------	---------------

height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Twitter share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```
.s7ecatalogviewer .s7twittershare {
  width:28px;
  height:28px;
}
.s7ecatalogviewer .s7twittershare[state='up'] {
  background-image:url(images/v2/TwitterShare_dark_up.png);
}
.s7ecatalogviewer .s7twittershare[state='over'] {
  background-image:url(images/v2/TwitterShare_dark_over.png);
}
.s7ecatalogviewer .s7twittershare[state='down'] {
  background-image:url(images/v2/TwitterShare_dark_down.png);
}
.s7ecatalogviewer .s7twittershare[state='disabled'] {
  background-image:url(images/v2/TwitterShare_dark_disabled.png);
}
```

View All Favorites button

The position of the button is fully managed by the Favorites menu.

The appearance of the View All Favorites button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7viewallfavoritebutton
```

CSS properties of the Remove Favorite button

background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
width	Width of the button.
height	Height of the button.



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the state when a user can add a new Favorite icon by clicking or tapping. *selected='false'* corresponds to the normal operation mode when a user can zoom, pan, and swap pages.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a View All favorites button that is 28 x 28 pixels, and displays a different image for each of the four different button states when selected or not selected.

```
.s7ecatalogviewer .s7viewallfavoritebutton {
  width:28px;
  height:28px;
}
.s7ecatalogviewer .s7viewallfavoritebutton[selected='false'][state='up'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_up.png);
}
.s7ecatalogviewer .s7viewallfavoritebutton[selected='false'][state='over'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_over.png);
}
.s7ecatalogviewer .s7viewallfavoritebutton[selected='false'][state='down'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_down.png);
}
.s7ecatalogviewer .s7viewallfavoritebutton[selected='false'][state='disabled'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_disabled.png);
}
.s7ecatalogviewer .s7viewallfavoritebutton[selected='true'][state='up'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_over.png);
}
.s7ecatalogviewer .s7viewallfavoritebutton[selected='true'][state='over'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_over.png);
}
.s7ecatalogviewer .s7viewallfavoritebutton[selected='true'][state='down'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_over.png);
}
.s7ecatalogviewer .s7viewallfavoritebutton[selected='true'][state='disabled'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_disabled.png);
}
```

Zoom in button

Clicking or tapping this button zooms in on an image in the main view. This button appears in the main control bar. This button is not displayed on mobile phones to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7zoominbutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.
left	Position from the left border of the main control bar, including padding.

CSS property	Description
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom in button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 103 pixels from the right edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7zoominbutton {
bottom:4px;
right:103px;
width:28px;
height:28px;
}
.s7ecatalogviewer .s7zoominbutton [state='up'] {
background-image:url(images/v2/ZoomInButton_dark_up.png);
}
.s7ecatalogviewer .s7zoominbutton [state='over'] {
background-image:url(images/v2/ZoomInButton_dark_over.png);
}
.s7ecatalogviewer .s7zoominbutton [state='down'] {
background-image:url(images/v2/ZoomInButton_dark_down.png);
}
.s7ecatalogviewer .s7zoominbutton [state='disabled'] {
background-image:url(images/v2/ZoomInButton_dark_disabled.png);
}
```

Zoom out button

Clicking or tapping this button zooms out on an image in the main view. This button does not display on mobile phones to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7zoomoutbutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.

CSS property	Description
right	Position from the right border of the main control bar, including padding.
left	Position from the left border of the main control bar, including padding.
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom out button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 75 pixels from the right edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7zoomoutbutton {
bottom:4px;
right:75px;
width:28px;
height:28px;
}
.s7ecatalogviewer .s7zoomoutbutton [state='up'] {
background-image:url(images/v2/ZoomOutButton_dark_up.png);
}
.s7ecatalogviewer .s7zoomoutbutton [state='over'] {
background-image:url(images/v2/ZoomOutButton_dark_over.png);
}
.s7ecatalogviewer .s7zoomoutbutton [state='down'] {
background-image:url(images/v2/ZoomOutButton_dark_down.png);
}
.s7ecatalogviewer .s7zoomoutbutton [state='disabled'] {
background-image:url(images/v2/ZoomOutButton_dark_disabled.png);
}
```

Zoom reset button

Clicking or tapping this button resets an image in the main view. This button appears in the main control bar on desktop systems and tablets. On mobile phones, this button shows in the bottom center over the image. However, it is not displayed when the image is in a reset state. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7zoomresetbutton
```

CSS property	Description
top	Position from the top border of the main control bar (on desktops and tablets) or viewer (on mobile phones), including padding.
right	Position from the right border of the main control bar (on desktops and tablets) or viewer (on mobile phones), including padding.
left	Position from the left border of the main control bar (on desktops and tablets) or viewer (on mobile phones), including padding.
bottom	Position from the bottom border of the main control bar (on desktops and tablets) or viewer (on mobile phones), including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom reset button that is 28 x 28 pixels, positioned (on desktop) 4 pixels from the bottom and 47 pixels from the right edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogviewer .s7zoomresetbutton {
bottom:4px;
right:47px;
width:28px;
height:28px;
}
.s7ecatalogviewer .s7zoomresetbutton [state='up'] {
background-image:url(images/v2/ZoomResetButton_dark_up.png);
}
.s7ecatalogviewer .s7zoomresetbutton [state='over'] {
background-image:url(images/v2/ZoomResetButton_dark_over.png);
}
.s7ecatalogviewer .s7zoomresetbutton [state='down'] {
background-image:url(images/v2/ZoomResetButton_dark_down.png);
}
.s7ecatalogviewer .s7zoomresetbutton [state='disabled'] {
background-image:url(images/v2/ZoomResetButton_dark_disabled.png);
}
```

Support for Adobe Analytics tracking

The eCatalog Viewer supports Adobe Analytics tracking out of the box.

Out-of-the-box tracking

The eCatalog Viewer supports Adobe Analytics tracking out-of-the-box. To enable tracking, pass the proper company preset name as `config2` parameter.

The viewer also sends a single tracking HTTP request to the configured Image Server with the viewer type and version information.

Custom tracking

To integrate with third-party analytics systems it is necessary to listen to the `trackEvent` viewer callback and process the `eventInfo` argument of the callback function as necessary. The following code is an example of such handler function:

```
var eCatalogViewer = new s7viewers.eCatalogViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Viewers/Pluralist",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  },
  "handlers": {
    "trackEvent": function(objID, compClass, instName, timeStamp, eventInfo) {
      //identify event type
      var eventType = eventInfo.split(",")[0];
      switch (eventType) {
        case "LOAD":
          //custom event processing code
          break;
        //additional cases for other events
      }
    }
  }
});
```

The viewer tracks the following SDK user events:

SDK user event	Sent when...
LOAD	viewer is loaded first.
SWAP	an asset is swapped in the viewer using the <code>setAsset()</code> API.
ZOOM	an image is zoomed.
PAN	an image is panned.
SWATCH	an image is change by clicking or tapping on a swatch.
PAGE	a current frame is changed in the main view.
ITEM	an info panel pop-up is activated.

SDK user event	Sent when...
HREF	a user navigates to a different page because of clicking the image map.

Localization of user interface elements

Certain content that the eCatalog Viewer displays is subject to localization, including zoom buttons, page change buttons, thumbnail button, full screen button, close button, and scroll bar buttons.

Every textual content in the viewer that can be localized is represented by a special Viewer SDK identifier called SYMBOL. Any SYMBOL has a default associated text value for the English locale ("en") supplied with the out-of-the-box viewer, and also may have user-defined values set for as many locales as needed.

When the viewer starts, it checks the current locale to see if there is a user-defined value for each supported SYMBOL in the locale. If there is, it uses the user-defined value; otherwise, it falls back to the out-of-the-box default text.

User-defined localization data can be passed to the viewer as a localization JSON object. Such an object contains the list of supported locales, SYMBOL text values for each locale, and the default locale.

An example of such localization object:

```
{
  "en": {
    "CloseButton.TOOLTIP": "Close",
    "ZoomInButton.TOOLTIP": "Zoom In"
  },
  "fr": {
    "CloseButton.TOOLTIP": "Fermer",
    "ZoomInButton.TOOLTIP": "Agrandir"
  },
  defaultLocale: "en"
}
```

In the example above, the localization object defines two locales ("en" and "fr") and provides localization for two user interface elements in each locale.

The web page code should pass such localization object to the viewer constructor as a value of `localizedTexts` field of the configuration object. An alternative option is to pass the localization object by calling `setLocalizedTexts(localizationInfo)` method.

The following SYMBOLs are supported (assuming `containerId` is theID of the viewer container):

SYMBOL	Tool tip for...
CloseButton.TOOLTIP	Close button.
ZoomInButton.TOOLTIP	Zoom in button.
ZoomOutButton.TOOLTIP	Zoom out button.
ZoomResetButton.TOOLTIP	Zoom reset button.
FullScreenButton.TOOLTIP_SELECTED	Full screen button in normal state.

SYMBOL	Tool tip for...
FullScreenButton.TOOLTIP_UNSELECTED	Full screen button in full screen state.
ScrollUpButton.TOOLTIP	Scroll up button.
ScrollDownButton.TOOLTIP	Scroll down button.
<containerId>_rightButton.PanRightButton.TOOLTIP	Large next page button.
<containerId>_leftButton.PanLeftButton.TOOLTIP	Large previous page button.
<containerId>_lastPageButton.PanRightButton.TOOLTIP	Last page button.
<containerId>_secondaryLastPageButton.PanRightButton.TOOLTIP	Last page button.
<containerId>_firstPageButton.PanLeftButton.TOOLTIP	First page button.
<containerId>_secondaryFirstPageButton.PanLeftButton.TOOLTIP	First page button.
<containerId>_toolBarRightButton.PanRightButton.TOOLTIP	Next page button.
<containerId>_toolBarLeftButton.PanLeftButton.TOOLTIP	Previous page button.
ThumbnailPageButton.TOOLTIP_SELECTED	Thumbnails button in thumbnails mode.
ThumbnailPageButton.TOOLTIP_UNSELECTED	Thumbnails button in normal mode.
CloseButton.TOOLTIP	Close button.
InfoPanelPopup.TOOLTIP_CLOSE	Info Panel close button.
SocialShare.TOOLTIP	Social share tool.
EmailShare.TOOLTIP	Email share button.
EmailShare.HEADER	Email dialog header.
EmailShare.TOOLTIP_HEADER_CLOSE	Email dialog box upper-right close button.
EmailShare.INVALID_ADDRESSSS	Error message displayed in case email address is malformed.

SYMBOL	Tool tip for...
EmailShare.TO	Label for the "To" input field.
EmailShare.TOOLTIP_ADD	Add Another Email Address button.
EmailShare.ADD	Add Another Email Address button.
EmailShare.FROM	From input field.
EmailShare.MESSAGE	Message input field.
EmailShare.TOOLTIP_REMOVE	Remove Email Address button.
EmailShare.CANCEL	Caption for the Cancel button.
EmailShare.TOOLTIP_CANCEL	Cancel button.
EmbedShare.ACTION	Caption for the Select All button.
EmbedShare.TOOLTIP_ACTION	Select All button.
EmailShare.CLOSE	Caption for the close button displayed in the bottom of dialog after form submission.
EmailShare.TOOLTIP_CLOSE	Close button that is displayed in the bottom of dialog after form submission.
EmailShare.ACTION	Caption for the form submission button.
EmailShare.TOOLTIP_ACTION	Form submission button.
EmailShare.SEND_SUCCESS	Confirmation message displayed when email was sent successfully.
EmailShare.SEND_FAILURE	Error message that is displayed when email was not sent successfully.
EmbedShare.TOOLTIP	Embed share button.
EmbedShare.HEADER	Embed dialog box header.

SYMBOL	Tool tip for...
EmbedShare.TOOLTIP_HEADER_CLOSE	Embed dialog box upper-right close button.
EmbedShare.DESRIPTION	Description of the embed code text.
EmbedShare.EMBED_SIZE	Label for the embed size combo box.
EmbedShare.CANCEL	Caption for the Cancel button.
EmbedShare.TOOLTIP_CANCEL	Cancel button.
EmbedShare.CUSTOM_SIZE	Text for the last "custom size" entry in the embed size combo box.
LinkShare.TOOLTIP	Link share button.
LinkShare.HEADER	Link dialog box header.
LinkShare.TOOLTIP_HEADER_CLOSE	Link dialog box upper-right close button.
LinkShare.DESRIPTION	Description of the share link.
LinkShare.CANCEL	Caption for the Cancel button.
LinkShare.TOOLTIP_CANCEL	Cancel button.
LinkShare.ACTION	Caption for the Select All button.
LinkShare.TOOLTIP_ACTION	Select All button.
FacebookShare.TOOLTIP	Facebook share button.
TwitterShare.TOOLTIP	Twitter share button.
Print.TOOLTIP	Print button.
Print.HEADER	Print dialog header.
Print.TOOLTIP_HEADER_CLOSE	Print dialog box top right close button.
Print.PRINT_RANGE	Label for the "Select Print Pages" section.

SYMBOL	Tool tip for...
Print.PRINT_RANGE_CURRENT	Caption for the "Current pages" radio button.
Print.PRINT_RANGE_FROM	Caption for the "Spread range from" radio button.
Print.PRINT_RANGE_TO	Caption for the "to" numeric picker.
Print.PRINT_RANGE_ALL	Caption for the "All pages" radio button.
Print.PAGE_HANDLING	Label for the "Page Handling" section.
Print.PAGE_HANDLING_ONE	Caption for the "1 page per sheet" radio button.
Print.PAGE_HANDLING_TWO	Caption for the "2 pages per sheet" radio button.
Print.CANCEL	Caption for the Cancel button.
Print.TOOLTIP_CANCEL	Cancel button.
Print.ACTION	Caption for the Send to print button
Print.TOOLTIP_ACTION	Send to print button.
FavoritesMenu.TOOLTIP	Favorites menu button.
AddFavoriteButton.TOOLTIP_SELECTED	"Add favorite" button in edit Favorites mode.
AddFavoriteButton.TOOLTIP_UNSELECTED	"Add favorite" button in normal mode.
RemoveFavoriteButton.TOOLTIP_SELECTED	"Remove favorite" button in edit Favorites mode.
RemoveFavoriteButton.TOOLTIP_UNSELECTED	"Remove favorite" button in normal mode.
ViewAllFavoriteButton.TOOLTIP_SELECTED	"View all favorites" button when Favorites view is active.

SYMBOL	Tool tip for...
<code>ViewAllFavoriteButton.TOOLTIP_UNSELECTED</code>	"View all favorites" button when Favorites view is inactive.
<code>FavoritesEffect.TOOLTIP</code>	Single favorite icon.
<code>MediaSet.LABEL_XX[_YY]</code>	<p>Page label that is generated by the viewer at load time.</p> <p>The name of that symbol is a template, where <code>XX</code> is a zero-based spread index in landscape orientation, and optional <code>YY</code> is a zero-based page index inside the spread targeted by <code>XX</code>.</p> <p>Applies only for the initially loaded asset; ignored if an asset is changed using the <code>setAsset()</code> API call.</p>
<code>MediaSet.LABEL_DELIM</code>	Character used as a page labels delimiter in case labels are defined for left and right pages within a spread.
<code>ScrollLeftRightButton.TOOLTIP_SELECTED</code>	Main control bar scroll left button.
<code>ScrollLeftRightButton.TOOLTIP_UNSELECTED</code>	Main control bar scroll right button.

Image map support

The eCatalog Viewer supports the rendering of image map icons above the main view.

The appearance of map icons is controlled through CSS as described in [Image map effect](#).

Image maps perform one of the following three actions: redirect to an external web page, Info panel pop-up activation, and internal hyperlinks.

Redirect to an external web page

The `href` attribute of the image map has a URL to the external resource, either specified explicitly, or wrapped into one of the supported JavaScript template functions: `loadProduct()`, `loadProductCW()`, and `loadProductPW()`.

The following is an example of a simple URL redirect:

```
href=http://www.adobe.com
```

In this example, the same URL is wrapped with the `loadProduct()` function:

```
href=javascript:loadProduct("http://www.adobe.com");void(0);
```

Be aware that when you add the JavaScript code into the `HREF` attribute of your image map, the code is run on the client's computer. Therefore, make sure that the JavaScript code is secure.

Info Panel Popup activation

To work with Info panels, an image map has the `ROLLOVER_KEY` attribute set. Also, set the `href` attribute at the same time, otherwise the external URL processing interferes with the Info panel pop-up activation.

Finally, be sure that the viewer configuration includes the appropriate values for `InfoPanelPopup.template` and, optionally, `InfoPanelPopup.infoServerUrl` parameters.



Note: Be aware that when you configure Info Panel Popup, the HTML code and JavaScript code passed to the Info Panel runs on the client's computer. Therefore, make sure that such HTML code and JavaScript code are secure.

Internal hyperlinks

Clicking on an image map performs an internal page swap inside the viewer. To use that feature, an `href` attribute in the image map has the following special format:

```
href=target:idx
```

where `idx` is a zero-based index of the catalog spread.

The following is an example of an `href` attribute for an image map that points to the 3D spread in the eCatalog:

```
href=target:2
```

Managing page labels

There are two places in the viewer user interface where page labels are shown: thumbnails mode and the table of contents drop-down.

There are three types of labels that can be defined:

- Labels defined by the author using SYMBOL localization mechanism.
- Labels defined by the author on the backend, inside Scene7 Publishing System.
- Labels automatically generated by the viewer.

SYMBOL-based labels are defined using `MediaSet.LABEL_XX[_YY]` and `MediaSet.LABEL_DELIM` SYMBOLs as described in [Localization of user interface elements](#). You can define such labels either for the entire ecatalog spread, in which case you should use short SYMBOL syntax (`MediaSet.LABEL_XX`). Or, specify it for each page individually using full SYMBOL syntax (`MediaSet.LABEL_XX_YY`).

When you define labels for both pages in the ecatalog spread, the viewer concatenates these labels into one string using `MediaSet.LABEL_DELIM` SYMBOL. SYMBOL-based labels take precedence over labels that are defined on the backend or automatically generated labels by the viewer.

Labels defined in Scene7 Publishing System are stored in the `UserData` record of individual page images. Same as with SYMBOL-based labels. That is, if both pages in the ecatalog spread have labels defined, they are concatenated using `MediaSet.LABEL_DELIM` SYMBOL in landscape mode. Scene7 Publishing System labels take precedence over automatically generated labels, but are overridden by SYMBOL-based labels.

Automatically generated labels are sequential numbers assigned to all pages in the e-catalog. Automatically generated labels are ignored for the given spread if it has SYMBOL-based labels defined or Scene7 Publishing System labels defined.

In the table of contents, it is possible to disable the display of automatically generated labels using `showdefault` parameter.

Full screen support

The viewer supports full screen operation mode.

On modern desktop browsers, except Internet Explorer 10 and older, and on some touch devices, the viewer uses "native" full screen mode. This mode means that the entire device screen is occupied by the viewer content.

On iOS devices and on older Internet Explorer browsers, the viewer uses "simulated" full screen mode instead. In this mode, the viewer simply resizes to take the full area of the web browser window. Also, the web browser Chrome and other windows are still visible on the screen.

An end user enters and leaves full screen mode by pressing the Full Screen button in the viewer user interface. When "native" full screen mode is used on desktop, it is also possible to exit it by pressing **Esc**.

Print feature

The viewer lets you output the catalog content to a printer.

The print feature is triggered by a dedicated button in the toolbar. Clicking the button lets the user choose a print range and the number of pages per sheet.

The quality of the print can be adjusted using the `printquality` configuration parameter. Note that setting `printquality` to values significantly higher than the default is not recommended. The reason is because it leads to very high memory consumption by the web browser on the client's system. Also, make sure that the maximum image response size set for your SPS company is larger than the configured `printquality` value.



Note: The Print feature is only available on desktop systems, except Internet Explorer 9.

Download

It is possible to download the electronic catalog as a PDF file using the "Download" button in the control bar.

The "Download" button is automatically available in the viewer user interface when the following occurs:

- An actual PDF file is present in the customer's company.
- The name matches the name of the e-catalog asset that is passed to the viewer and includes a `.pdf` extension.
- The PDF file is published in SPS (Scene7 Publishing System).

Favorites feature

An end user can use the Favorites feature to mark products of interest directly in the e-catalog by adding "heart" icons to the e-catalog images. The viewer stores all the favorites on the client's system and displays them again when the same user visits the e-catalog again.

In addition, the viewer lets you review all favorites added to the e-catalog as a grid of thumbnail images.

All Favorites tools are grouped under one drop-down list in the control bar.

Viewer SDK namespace

The viewer is built of many Viewer SDK components. In most cases, the web page does not need to interact with SDK components API directly; all common needs are covered in the viewer API itself.

However, some advanced use cases require that the web page obtain a reference to an inner SDK component using the `getComponent()` viewer API and then use all the flexibility of the APIs of SDK itself.

The namespace that is used to load and initialize SDK components by the viewer depends on the environment in which the viewer is operating. If the viewer is running in AEM (Adobe Experience Manager), the viewer loads SDK components into `s7viewers.s7sdk` namespace. And the viewer served from Scene7 Publishing System loads the SDK into `s7classic.s7sdk`.

In either case, the namespace used by the SDK inside the viewer has either `s7viewers` or `s7classic` as the prefix. And, it is different from the plain `s7sdk` namespace used in the SDK User Guide or SDK API documentation.

For that reason it is important to use a fully qualified SDK namespace when you write custom application code that communicates with internal viewer components.

For example, if you plan to listen to `StatusEvent.NOTF_VIEW_READY` event and the viewer is served from Scene7 Publishing System, the fully qualified event type is `s7classic.s7sdk.event.StatusEvent.NOTF_VIEW_READY`, and the event listener code looks similar to the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var pageView = <instance>.getComponent("pageView");
    pageView.addEventListener(s7classic.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});
```

eCatalog Search

eCatalog Search Viewer is a catalog viewer that displays electronic brochures in a spread by spread or page by page manner. The eCatalog lets users navigate through the catalog using additional user interface elements or dedicated thumbnails mode. Users can also zoom in on every page for greater detail.

This viewer works with ecatalogs and supports optional image maps and social sharing tools. It has zoom tools, catalog navigation tools, full screen support, thumbnails, and an optional close button. The viewer also supports social sharing tools, Print, Download, and Favorites. It is designed to work on desktops and mobile devices.

The user can also do a keyword-based or phrase-based search over the catalog contents.



Note: Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.

Viewer type 513.

See [System requirements](#).

Demo URL

<https://www.adobe.com/uk/scene7/scene7-viewers/scene7-viewer-513.html>

Using eCatalog Viewer

eCatalog Search Viewer represents a main JavaScript file and a set of helper files (a single JavaScript include with all Viewer SDK components used by this particular viewer, assets, CSS) downloaded by the viewer in runtime

You can use eCatalog Search Viewer in pop-up mode using a production-ready HTML page provided with IS-Viewers or in embedded mode, where it is integrated into target web page using documented API.

Configuration and skinning are similar to that of the other viewers. All skinning is achieved by way of custom CSS.

See [Command reference common to all viewers – Configuration attributes](#) and [Command reference common to all Viewers – URL](#)

Interacting with eCatalog Search Viewer

eCatalog Search Viewer supports the following touch gestures that are common in other mobile applications.

Gesture	Description
Single tap	Selects new thumbnail in swatches.
Double tap	Zooms in one level until maximum magnification is reached. The next double tap gesture resets the viewer to the initial viewing state.
Pinch	Zooms in or out.
Horizontal swipe or flick	Scrolls through the list of catalog pages if a slide frame transition is used.
Vertical swipe or flick	When the image is in a reset state, it performs a native page scroll. When thumbnails are active, it scrolls the list of thumbnails.

It is possible to enable a realistic page flip animation effect for navigating between catalog pages. In such cases, a user can hold and drag a page corner and flip the page.

This viewer also supports both touch input and mouse input on Windows devices with a touch screen and mouse. This support, however, is limited to Chrome, Internet Explorer 11, and Edge web browsers only.

Social media sharing tools with eCatalog Search Viewer

The eCatalog Search Viewer supports social sharing tools. They are available as a button in the main control bar which expands into a sharing tool bar when a user clicks or taps on it.

The sharing tool bar contains icons for each type of sharing channel supported which includes Facebook, Twitter, email share, embed code share, and link share. When email share, embed share, or link share tools are activated, the viewer displays a modal dialog box with a corresponding data entry form. When Facebook or Twitter is called, the viewer redirects the user to a standard sharing dialog from a social service. Sharing tools are not available in full screen mode because of web browser security restrictions.

The viewer's Search feature is available as a looking glass icon in the main toolbar. Clicking or tapping the icon activates the Search panel with an input field. After entering a keyword or phrase and pressing Enter, the viewer renders search results in the panel and highlights found words in the main view.

Embedding eCatalog Search Viewer

Different web pages have different needs for viewer behavior. Sometimes a web page provides a link that, when clicked, opens the viewer in a separate browser window. In other cases, it is necessary to embed the viewer right in the hosting page. In the latter case, the web page may have a static page layout, or use responsive design that displays differently on different devices or for different browser window sizes. To accommodate these needs, the viewer supports three primary operation modes: pop-up, fixed size embedding, and responsive design embedding.

About pop-up mode

In pop-up mode, the viewer is opened in a separate web browser window or tab. It takes the entire browser window area and adjusts in case the browser is resized, or a mobile device's orientation is changed.

Pop-up mode is the most common for mobile devices. The web page loads the viewer using `window.open()` JavaScript call, properly configured A HTML element, or any other suitable method.

It is recommended that you use an out-of-the-box HTML page for pop-up operation mode. In this case, it is called `eCatalogSearchViewer.html` and is located within the `html5/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/eCatalogSearchViewer.html
```

You can achieve visual customization by applying custom CSS.

The following is an example of HTML code that opens the viewer in a new window:

```
<a href="https://arxiv.org/abs/1908.07444v1" target="_blank">Open pop-up viewer</a>
```

About fixed size embedding mode and responsive design embedding mode

In the embedded mode, the viewer is added to the existing web page, which may already have some customer content not related to the viewer. The viewer normally occupies only a part of a web page's real estate.

The primary use cases are web pages oriented for desktops or tablet devices, and also responsive designed pages that adjust layout automatically depending on the device type.

Fixed size embedding is used when the viewer does not change its size after initial load. This is the best choice for web pages that have a static layout.

Responsive design embedding assumes that the viewer may need to resize at runtime in response to the size change of its container `DIV`. The most common use case is adding a viewer to a web page that uses a flexible page layout.

In responsive design embedding mode, the viewer behaves differently depending on the way web page sizes its container `DIV`. If the web page sets only the width of the container `DIV`, leaving its height unrestricted, the viewer automatically chooses its height according to the aspect ratio of the asset that is used. This functionality ensures that the asset fits perfectly into the view without any padding on the sides. This use case is the most common for web pages using responsive layout frameworks like Bootstrap, Foundation, and so on.

Otherwise, if the web page sets both the width and the height for the viewer's container `DIV`, the viewer fills just that area and follows the size that the web page layout provides. A good example is embedding the viewer into a modal overlay, where the overlay is sized according to web browser window size.

Fixed size embedding

You add the viewer to a web page by doing the following:

1. Adding the viewer JavaScript file to your web page.

2. Defining the container DIV.
3. Setting the viewer size.
4. Creating and initializing the viewer.
1. Adding the viewer JavaScript file to your web page.

Creating a viewer requires that you add a script tag in the HTML head. Before you can use the viewer API, be sure that you include `eCatalogSearchViewer.js`. The `eCatalogSearchViewer.js` file is located under the `html5/js/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/js/eCatalogSearchViewer.js
```

You can use a relative path if the viewer is deployed on one of the Adobe Scene7 servers and it is served from the same domain. Otherwise, you specify a full path to one of Adobe Scene7 servers that have the IS-Viewers installed.

The relative path looks like the following:

```
<script language="javascript" type="text/javascript"
src="/s7viewers/html5/js/eCatalogSearchViewer.js"></script>
```

2. Defining the container DIV.

Add an empty DIV element to the page where you want the viewer to appear. The DIV element must have its ID defined because this ID is passed later to the viewer API.

The placeholder DIV is a positioned element, meaning that the `position` CSS property is set to `relative` or `absolute`.

The following is an example of a defined placeholder DIV element:

```
<div id="s7viewer" style="position:relative"></div>
```

3. Setting the viewer size

You can set the static size for the viewer by either declaring it for `.s7ecatalogsearchviewer` top-level CSS class in absolute units, or by using `stagesize` modifier.

You can put sizing in CSS directly on the HTML page, or in a custom viewer CSS file, which is then later assigned to a viewer preset record in Scene7 Publishing System, or passed explicitly using a `style` command.

See [Customizing eCatalog Viewer](#) for more information about styling the viewer with CSS.

The following is an example of defining a static viewer size in HTML page:

```
#s7viewer.s7ecatalogsearchviewer {
width: 640px;
height: 480px;
}
```

You can set the `stagesize` modifier either in the viewer preset record in Scene7 Publishing System, or pass it explicitly with the viewer initialization code with `params` collection, or as an API call as described in the Command Reference section, like the following:

```
eCatalogSearchViewer.setParam("stagesize",
"640,480");
```

4. Initializing the viewer.

When you have completed the steps above, you create an instance of `s7viewers.eCatalogSearchViewer` class, pass all configuration information to its constructor and call `init()` method on a viewer instance. Configuration information is passed to the constructor as a JSON object. At minimum, this object has the `containerId` field which holds the name of

viewer container ID and nested params JSON object with configuration parameters supported by the viewer. In this case, the params object must have at least the Image Serving URL passed as `serverUrl` property, and the initial asset as `asset` parameter. JSON-based initialization API lets you create and start the viewer with single line of code.

It is important to have the viewer container added to the DOM so that the viewer code can find the container element by its ID. Some browsers delay building DOM until the end of the web page. For maximum compatibility, however, call the `init()` method just before the closing BODY tag, or on the `body onload()` event.

At the same the container element should not necessarily be part of the web page layout just yet. For example, it may be hidden using `display:none` style assigned to it. In this case, the viewer delays its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load automatically resumes.

The following is an example of creating a viewer instance, passing the minimum necessary configuration options to the constructor, and calling the `init()` method. The example assumes `eCatalogSearchViewer` is the viewer instance; `s7viewer` is the name of placeholder DIV; `http://s7dl.scene7.com/is/image/` is the Image Serving URL, and `Viewers/Pluralist` is the asset:

```
<script type="text/javascript">
var eCatalogSearchViewer = new s7viewers.eCatalogSearchViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Viewers/Pluralist",
    "serverurl": "http://s7dl.scene7.com/is/image/",
    "searchserverurl": "http://s7search1.scene7.com/s7search/"
  }
}).init();
</script>
```

The following code is a complete example of a trivial web page that embeds the eCatalog Search Viewer with a fixed size:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/eCatalogSearchViewer.js"></script>
<style type="text/css">
#s7viewer.s7ecatalogsearchviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var eCatalogSearchViewer = new s7viewers.eCatalogSearchViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Viewers/Pluralist",
    "serverurl": "http://s7dl.scene7.com/is/image/",
    "searchserverurl": "http://s7search1.scene7.com/s7search/"
  }
}).init();
</script>
</body>
</html>
```

Responsive design embedding with unrestricted height

With responsive design embedding, the web page normally has some kind of flexible layout in place that dictates the runtime size of the viewer's container DIV. For purposes of this example, assume that the web page allows the viewer's container DIV to

take 40% of the web browser window size, leaving its height unrestricted. The resulting web page HTML code looks like the following:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
  .holder {
    width: 40%;
  }
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>
```

Adding the viewer to such a page is similar to fixed size embedding, with the only difference being that you do not need to explicitly define viewer size.

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Creating and initializing the viewer.

All the steps above are the same as with fixed size embedding. Add the container DIV to the existing "holder" DIV. The following code is a complete example. You can see how the viewer size changes when the browser is resized, and how the viewer aspect ratio matches the asset.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/eCatalogSearchViewer.js"></script>
<style type="text/css">
  .holder {
    width: 40%;
  }
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var eCatalogSearchViewer = new s7viewers.eCatalogSearchViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Viewers/Pluralist",
    "serverurl":"http://s7dl.scene7.com/is/image/",
    "searchserverurl":"http://s7search1.scene7.com/s7search/"
  }
}).init();
</script>
</body>
</html>
```

The following examples page illustrates more real-life use cases of responsive design embedding with unrestricted height:

https://marketing.adobe.com/resources/help/en_US/s7/vlist/vlist.html

Flexible size embedding with width and height defined

In case of flexible-size embedding with width and height defined, the web page styling is different. That is, it provides both sizes to the "holder" DIV and centers it in the browser window. Also, the web page sets the size of the HTML and BODY element to 100%:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>
```

The remaining embedding steps are identical to responsive design embedding with unrestricted height. The resulting example is the following:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/eCatalogSearchViewer.js"></script>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var eCatalogSearchViewer = new s7viewers.eCatalogSearchViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Viewers/Pluralist",
    "serverurl":"http://s7dl.scene7.com/is/image/",
    "searchserverurl":"http://s7search1.scene7.com/s7search/"
  }
}).init();
</script>
</body>
</html>
```

Embedding Using Setter-based API

Instead of using JSON-based initialization it is possible to use setter-based API and no-args constructor. With that API constructor does not take any parameters and configuration parameters is specified using `setContainerId()`, `setParam()`, and `setAsset()` API methods with separate JavaScript calls.

The following example shows fixed size embedding with setter-based API:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/eCatalogSearchViewer.js"></script>
<style type="text/css">
#s7viewer.s7ecatalogsearchviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var eCatalogSearchViewer = new s7viewers.eCatalogSearchViewer();
eCatalogSearchViewer.setContainerId("s7viewer");
eCatalogSearchViewer.setParam("serverurl", "http://s7dl.scene7.com/is/image/");
eCatalogSearchViewer.setParam("searchserverurl", "http://s7search1.scene7.com/s7search/");
eCatalogSearchViewer.setAsset("Viewers/Pluralist");
eCatalogSearchViewer.init();
</script>
</body>
</html>
```

Command reference – Configuration attributes

Configuration attributes documentation for eCatalog Viewer.

Any configuration command can be set in URL or using `setParam()`, or `setParams()`, or both, API methods. You can also specify any configuration attribute that is specified in the server-side configuration record.

For some configuration commands you can prefix them with the class name or instance name of corresponding Viewer SDK component. An instance name of the component is dynamic and depends on the ID of the viewer container DOM element passed to `setContainerId()` API method. Documentation includes optional prefix for such commands. For example, `zoomstep` command is documented as follows:

```
[PageView.|<containerId>_pageView].zoomstep
```

which means that you can use this command as:

- `zoomstep` (short syntax)
- `PageView.zoomstep` (qualified with component class name)
- `cont_pageView.zoomstep` (qualified with component ID, assuming `cont` is the ID of the container element)

See also [Command reference common to all viewers – Configuration attributes](#)

Closebutton

```
closebutton=0|1
```

0-1	Set to 1 to enable display of the close button. Or, set to 0 to hide the close button.
-----	--

	The close button is supported only on touch devices; it cannot be displayed on desktop systems.
--	---

Properties

Optional.

Default

0

Example

```
closebutton=1
```

ControlBar.transition

[ControlBar. |<containerId>_controls.]transition=none|fade[,delaytohide[,duration]

none fade	Specifies the effect type that is used to show or hide the control bar and its content. Use none for instant show and hide; fade provides a gradual fade in and fade out effect (not supported on Internet Explorer 8).
delaytohide	Specifies the time in seconds between the last mouse/touch event that the control bar registers and the time control bar hides. If set to -1 the component never triggers its auto-hide effect and always stay visible on the screen.
duration	Sets the duration of the fade in and fade out animation, in seconds.

Properties

Optional. This command is ignored on touch devices, where the control bar auto-hide is disabled.

Default

fade,2,0.5

Example

```
transition=none
```

direction

direction=auto|left|right

auto left right	Specifies the way pages are displayed in the main view and thumbnails. It also specifies the way the user interacts with the viewer user interface to change between catalog frames.
-----------------	--

When `left` is used it sets a right alignment for the initial page, and left alignment for the last page. It stitches individual page sub-images for left-to-right rendering order. It also sets the main view to use right-to-left slide animation to advance the catalog (in case `PageView.frametransition` is set to `slide`). Finally, thumbnails are set for a left-to-right fill order.

Similarly, when `right` is used it sets a left alignment for the initial page, and right alignment for the last page. It stitches individual page sub-images for right-to-left rendering order. It also sets the main view to use left-to-right slide animation to advance the catalog (in case `PageView.frametransition` is set to `slide`). Finally, it reverses the thumbnails order so that the thumbnails view is filled in right-to-left, top-to-bottom direction.

When `auto` is set, the viewer applies `right` mode when locale is set to `ja`; otherwise, it uses `left` mode.

Properties

Optional.

Default

`auto`

Example

`direction=right`

EmailShare.emailurl

```
[EmailShare.|<containerId>_emailShare.]emailurl=emailurl
```

<i>emailurl</i>	Specifies the base URL for Scene7 OnDemand email service.
-----------------	---

Properties

Optional.

Default

`/s7/emailFriend`

Example

`emailurl=http://s7d1.scene7.com/s7/emailFriend`

EmbedShare.embedsizes

```
[EmbedShare.|<containerId>_embedShare.]embedsizes=width,height[,0|1][;width,height[,0|1]]
```

<i>width</i>	Embed width.
--------------	--------------

<i>height</i>	Embed height.
0 1	Specifies whether this list item should be initially preselected in the combo box.

Properties

Optional.

Default

1280,960;640,480;320,240

Example

embedsizes=800,600;640,480,1

FavoritesEffect.expiration

[FavoritesEffect. | <containerId>_favoritesEffect.]expiration=days

<i>days</i>	Number of days the collection of favorites is kept on the client's system before they expire. Every time a user visits the catalog and makes a change to the favorites, such as adding or removing, the expiration timer is reset.
-------------	--

Properties

Optional.

Default

30

Example

expiration=7

FavoritesMenu.bearing

Specifies the direction of slide animation for the buttons container.

[FavoritesMenu. | <containerId>_favoritesMenu.]bearing=up|down|left|right|fit-vertical|fit-lateral

up down left right fit-vertical fit-lateral	<p>When set to up, down, left, or right, the panel rolls out in specified direction without an additional bounds check, which results in panel clipping by an outside container.</p> <p>When set to fit-vertical, the component first shifts the base panel position to the bottom of the Favorites menu and tries to roll out the panel in one of the following directions from such base location: bottom, right, left. With each attempt the component checks if the panel is clipped by an outside container. If all attempts fail, the component tries to shift the base panel position to the top and repeat roll out attempts from a top, right, and left direction.</p>
---	---

	When set to <code>fit-lateral</code> , the component uses a similar logic. The base is shifted to the right first, trying right, down, and up roll out directions. Then, it shifts the base to the left, trying left, down and up roll out directions.
--	--

Properties

Optional.

Default

`fit-vertical`

Example

`bearing=left`

FavoritesView.align

`[FavoritesView. | <containerId>_favoritesView.]align=left | center | right , top | center | bottom`

<code>left center right , top center bottom</code>	<p>Specifies the internal horizontal alignment—or anchoring—of the thumbnails container within the component area.</p> <p>In FavoritesView, the internal thumbnail container is sized so that only a whole number of thumbnails are shown. As a result, there is some padding between the internal container and the external component bounds.</p> <p>This modifier specifies how the internal thumbnails container is positioned horizontally inside the component.</p>
--	---

Properties

Optional.

Default

`center , center`

Example

`align=left , top`

FavoritesView.favoritesThumbView

`[FavoritesView. | <containerId>_favoritesView.]favoritesThumbArea=area`

<code>area</code>	<p>Specifies the crop area for Favorites thumbnail. Expressed as a relative value to the total frame size, with a range from 0 to 1 . 0.</p> <p>A value of 1 means that the entire frame image is used for the thumbnail.</p> <p>A value of 0 . 1 means only 10% of the frame size is used.</p>
-------------------	---

Properties

Optional.

Default

0.1

Example

favoritesThumbArea=0.5

FavoritesView.fmt

[FavoritesView. | <containerId>_favoritesView.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha

jpg jpeg png png-alpha gif gif-alpha	Specifies the image format used by the component for loading images from Image Server. The format is any value that is supported by the Image Server and the client browser. If the image format ends with -alpha, the component renders the images as transparent content. For all other image format values, the component treats images as opaque.
--------------------------------------	--

Properties

Optional.

Default

jpeg

Example

fmt=png-alpha

FavoritesView.iscommand

The Image Serving command string that is applied to all thumbnails.

[FavoritesView. | <containerId>_favoritesView.]iscommand=*isCommand*

<i>isCommand</i>	If specified in the URL, all occurrences of & and = must be HTTP-encoded as %26 and %3D, respectively.
------------------	--

Properties

Optional.

Default

None.

Example

When specified in the viewer URL.

iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000

When specified in the config data.

iscommand=op_sharpen=1&op_colorize=0xff0000

FavoritesView.maxloadradius

[FavoritesView. | <containerId>_favoritesView.]maxloadradius=-1 | 0 | *preloadnbr*

<i>preloadnbr</i>	<p>Specifies the component preload behavior.</p> <p>When set to -1, all thumbnails are loaded simultaneously when the component is initialized or the asset is changed.</p> <p>When set to 0, only visible thumbnails are loaded.</p> <p>When set to <i>preloadnbr</i>, you can specify how many invisible rows around visible area are preloaded.</p>
-------------------	--

Properties

Optional.

Default

1

Example

maxloadradius=0

FavoritesView.textpos

[FavoritesView. | <containerId>_favoritesView.]textpos=bottom | top | left | right | none | tooltip

bottom top left right none tooltip	<p>Specifies where the label is drawn relative to the thumbnail image. That is, the label is centered at the specified location relative to the thumbnail.</p> <p>When <i>tooltip</i> is specified, the label text is displayed as a floating tooltip over the thumbnail image.</p> <p>When set to <i>none</i>, it turns label display off.</p>
--	---

Properties

Optional.

Default

bottom

Example

textpos=top

ImageMapEffect.mapTips

[ImageMapEffect . | <containerId>_imageMapEffect .]mapTips=0|1

0 1	Specifies whether tool tips are enabled for individual map area elements. Ignored on touch devices, including touch-enabled desktop systems.
-----	---

Properties

Optional.

Default

0

Example

mapTips=1

ImageMapEffect.mode

[ImageMapEffect . | <containerId>_imageMapEffect .]mode=icon|region|auto|none

icon region auto none	Specifies the appearance of the image map. <ul style="list-style-type: none">• icon map icons are statically shown on the desktop and touch devices.• region renders image map regions; on desktop, they are shown on roll over and on touch devices they are always visible.• auto on desktop systems, image map regions are shown on roll over and on touch devices map icons are always visible.• none disables image maps.
-----------------------	---

Properties

Optional.

Default

icon

Example

mode=auto

ImageMapEffect.rollover

```
[ImageMapEffect. | <containerId>_imageMapEffect. ]rollover=0|1
```

0 1	<p>Specifies when to display the info panel.</p> <p>If set to 1, the info panel is displayed when the mouse enters image map area (in case the image map has non-empty, rollover_key attribute).</p> <p>If set to 0 info panel is triggered when the image map is clicked (if the image map has a non-empty rollover_key and empty href attributes).</p> <p>Ignored on touch devices, including touch-enabled desktop systems, and is automatically set to 0.</p>
-----	---

Properties

Optional.

Default

0

Example

```
rollover=1
```

InfoPanelPopup.infoServerUrl

```
[InfoPanelPopup. | <containerId>_infoPanelPopup. ]infoServerUrl=infoserverurl
```

infoserverurl	<p>Info server URL template is used to fetch key/value pairs for the variable substitution in the info panel content template. The specified template typically contains macro place holders that are replaced with the actual data before the request is sent to the server.</p> <p>\$1\$ is replaced with the rollover value that triggered the InfoPanelPopup activation.</p> <p>\$2\$ is replaced with the sequence number of the current frame in the image set.</p> <p>\$3\$ is replaced with the first path element specified in the name of the parent set of the current item. It typically corresponds to the catalog id.</p> <p>\$4\$ is replaced with the following element in the path and corresponds to the asset id. The actual info server request syntax is info server dependent and it differs from server to server. For example, the following is a typical Scene7 info server request template:</p> <pre>http://server_domain/s7info/s7/\$3\$/\$4\$/\$1\$</pre>
---------------	--



Note: Be aware that when you configure Info Panel Popup, the HTML code and JavaScript code passed to the Info Panel runs on the client's computer. Therefore, make sure that such HTML code and JavaScript code are secure.

Properties

Optional.

Default

None.

Example

infoServerUrl= http://s7info1.scene7.com/s7info/s7/\$3\$/\$4\$/\$1\$

InfoPanelPopup.showhidetransition

[InfoPanelPopup. | <containerId>_infoPanelPopup.]showhidetransition=fade|none[, time]

fade none	Specifies the type of info panel show/hide animation.
time	Duration (in seconds) for the show or hide animation.

Properties

Optional.

Default

fade,0.3

Example


showhidetransition=none

InfoPanelPopup.template

[InfoPanelPopup. | <containerId>_infoPanelPopup.]template=template

template	<p>The content template that the data returned from the info server is merged into.</p> <p>The content template is an XML following this DTD:</p> <pre><!DOCTYPE info [<!ELEMENT info (var #PCDATA) <!ELEMENT var (#PCDATA)> <!ATTLIST var name CDATA #REQUIRED rollover CDATA #IMPLIED > ></pre> <p>The actual syntax for the content template is the following:</p> <pre><info> <var name='VAR_NAME' rollover='ROLLOVER_KEY'><![CDATA[VAR_VALUE]]> <![CDATA[TEMPLATE_CONTENT]]> </info></pre>
----------	---

	<p>That is, the template must start with the <code><info></code> element that may contain optional default <code><var></code> elements. The template content itself, <code>TEMPLATE_CONTENT</code> is HTML text. In addition, the content template may contain variable names enclosed in <code>\$</code> characters. Those characters are replaced with the variable values that the info server returns or with default ones.</p> <p>Default variables that are defined in the template can be either global (if rollover attribute is not set) or specific to a certain rollover key (if rollover attribute is present).</p> <p>During template processing variables that are specific to roll over keys take precedence over global variables.</p>
--	--

 **Note:** Be aware that when you configure Info Panel Popup, the HTML code and JavaScript code passed to the Info Panel runs on the client's computer. Therefore, make sure that such HTML code and JavaScript code are secure.

Properties

Optional.

Default

None.

Example

Assuming that the info server response returns the product name as variable `1` and product image URL is returned as variable `2`.

```
template=<info><![CDATA[Product description:$1$<br>Product image:]]></info>
```

InitialFrame

```
initialFrame=frame
```

<code>frame</code>	Specifies a zero-based spread index to display on viewer load. The index matches the index of the spread in landscape mode. If the viewer is rotated to portrait, the viewer displays the leftmost page from the spread that is pointed to by <code>frameIdx</code> .
--------------------	---

Properties

Optional.

Default

0

Example

When specified in the viewer URL.

```
initialFrame=2
```

PageView.doubleclick

[PageView. | <containerId>_pageView.].doubleclick=*none* | *zoom* | *reset* | *zoomReset*

<i>none</i> <i>zoom</i> <i>reset</i> <i>zoomReset</i>	Configures the mapping of double-click/tap to zoom actions. Setting to <i>none</i> disables double-click/tap zoom. If set to <i>zoom</i> clicking the image zooms in one zoom step; CTRL+Click zooms out one zoom step. Setting to <i>reset</i> causes a single click on the image to reset the zoom to the initial zoom level. For <i>zoomReset</i> , reset is applied if the current zoom factor is at or beyond the specified limit, otherwise zoom is applied.
---	--

Properties

Optional.

Default

reset on desktop computers; *zoomReset* on touch devices.

Example

doubleclick=zoom

PageView.enableHD

[PageView. | <containerId>_pageView.].enableHD=*always* | *never* | *limit*[, *number*]

<i>always</i> <i>never</i> <i>limit</i>	Enable, limit or disable optimization for devices where <i>devicePixelRatio</i> is greater than 1, that is devices with high-density display like iPhone4 and similar devices. If active then the component limits the size of the IS image request as if the device only had a pixel ratio of 1 and that way reducing the bandwidth. See example below
<i>number</i>	If using the <i>limit</i> setting, the component enables high pixel density only up to the specified limit. See example below.

Properties

Optional.

Default

limit, 1500

Example

The following are the expected results when you use this configuration attribute with the viewer, and the viewer size is 1000 x 1000:

If the set variable equals	Result
always	<p>The pixel density of the screen/device is always taken into account.</p> <ul style="list-style-type: none"> • If the screen pixel density = 1, then the requested image is 1000 x 1000. • If the screen pixel density = 1.5, then the requested image is 1500 x 1500. • If the screen pixel density = 2, then the requested image is 2000 x 2000.
never	<p>It always uses a pixel density of 1 and ignores the device's HD capability. Therefore, the requested image requested is always 1000 x 1000.</p>
limit<number>	<p>A device pixel density is requested and served only if the resulting image is below the specified limit.</p> <p>The limit number applies to either the width or the height dimension.</p> <ul style="list-style-type: none"> • If the limit number is 1600, and the pixel density is 1.5, then the 1500 x 1500 image is served. • If the limit number is 1600, and the pixel density is 2, then the 1000 x 1000 image is served because the 2000 x 2000 image exceeds the limit. <p>Best practice: The limit number needs to work in conjunction with the company setting for maximum size image. Therefore, set the limit number to equal the company maximum image size setting.</p>

PageView.fmt

[PageView. | <containerId>_pageView.]fmt=jpg | jpeg | png | png-alpha | gif | gif-alpha

jpg jpeg png png-alpha gif gif-alpha	<p>Specifies the image format that the component uses for loading images from Image Server. If the specified format ends with <code>-alpha</code>, the component renders images as transparent content. For all other image formats the component treats images as opaque. The component has a white background by default. Therefore, to make it transparent, set the <code>background-color</code> CSS property to <code>transparent</code>.</p>
--	--

Properties

Optional.

Default


jpeg

Example

fmt=png-alpha

PageView.frametransition

```
[PageView. | <containerId>_pageView.]frametransition=slide|turn|auto[,duration]
```

slide turn auto	<p>Specifies the type of effect that is applied on frame change.</p> <ul style="list-style-type: none"> • <code>slide</code> activates a transition where the old frame slides out of view and the new frame slides in to view. • <code>turn</code> enables a page flip effect, when a user can drag one of the four spread corners and perform an interactive page flip. <p>When <code>turn</code> is used the appearance of the component is controlled with the <code>pageturnstyle</code> modifier and the <code>.s7pagedivider</code> CSS class is ignored.</p> <p> Note:</p> <p><i>turn animation is not supported on Motorola Xoom.</i></p> <ul style="list-style-type: none"> • <code>auto</code> sets the turn frame transition on desktop systems and the slide transition on touch devices.
duration	Specifies the duration in seconds of a <code>slide</code> or <code>turn</code> transition effect.

Properties

Optional.

Default

slide,0.3

Example

```
frametransition=auto,1
```

PageView.iconEffect

```
[PageView. | <containerId>_pageView.]iconeffect=0|1[,count][,fade][,autoHide]
```

0 1	Enables the <code>iconeffect</code> to display on the top of the image when the image is in a reset state and it is suggestive of an available action to interact with the image.
count	Specifies the maximum number of times the <code>iconeffect</code> appears and reappears. A value of <code>-1</code> indicates that the icon always reappears indefinitely.
fade	Specifies the duration of the show or hide animation, in seconds.

<i>autoHide</i>	Sets the number of seconds that the <code>iconEffect</code> stays fully visible before it auto hides. That is, the time after the fade-in animation is completed but before the fade out animation starts. A setting of 0 disables the auto-hide behavior.
-----------------	--

Properties

Optional.

Default


1,1,0.3,3

Example

`iconEffect=0`

PageView.iscommand

`[PageView. | <containerId>_pageView.]iscommand=isCommand`

<i>isCommand</i>	<p>The Image Serving command string that is applied to page image. If specified in the URL all occurrences of & and = must be HTTP-encoded as %26 and %3D, respectively.</p> <p> Note: Image sizing manipulation commands are not supported.</p>
------------------	--

Properties

Optional.

Default

None.

Example

When specified in the viewer URL.

`iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000`

When specified in the config data.

`iscommand=op_sharpen=1&op_colorize=0xff0000`

PageView.maxloadradius

`[PageView. | <containerId>_pageView.]maxloadradius=-1 | 0 | preloadnbr`

<code>-1 0 preloadnbr</code>	<p>Specifies the component preload behavior.</p> <p>When set to -1 the component preloads all the catalog frames when in an idle state.</p>
----------------------------------	---

	<p>When set to 0 the component loads only the frame that is currently visible, previous and next frame.</p> <p>Set <i>preloadnbr</i> to define how many invisible frames around the currently displayed frame are preloaded in an idle state.</p>
--	---

Properties

Optional.

Default

1

Example

maxloadradius=0

PageView.pageturnstyle

[PageView.<containerId>_pageView.]pageturnstyle=dividerWidth,dividerColor,dividerOpacity,borderOnOff,borderColor,fillColor

Controls the component appearance when a PageView.frametransition is set to turn or to auto on desktop systems.

<i>dividerWidth</i>	The width in pixels of the page divider shadow that separates the left and right pages in the spread. It also controls the width of the running shadow displayed next to the turning page.
<i>dividerOpacity</i>	The shadow color in RRGGBB format.
<i>dividerOpacity</i>	The shadow opacity in the range of 0 to 1.
<i>borderOnOff</i>	The flag (either 0 or 1) which turns the border around the turning page on and off.
<i>borderColor</i>	The border color in RRGGBB format.
<i>fillColor</i>	The color of the solid fill of the component area used during page turn animation, in RRGGBB format.

Properties

Optional.

Default

40,909090,1,1,909090,FFFFFF

Examples

pageturnstyle=20,FF0000,1,1,00FF00,0000FF

PageView.singleclick

[PageView. | <containerId>_pageView.]singleclick=*none* | *zoom* | *reset* | *zoomReset*

<i>none</i> <i>zoom</i> <i>reset</i> <i>zoomReset</i>	Configures the mapping of single-click/tap to zoom actions. Setting to <i>none</i> disables single-click/tap zoom. If set to <i>zoom</i> clicking the image zooms in one zoom step; CTRL+Click zooms out one zoom step. Setting to <i>reset</i> causes a single click on the image to reset the zoom to the initial zoom level. For <i>zoomReset</i> , reset is applied if the current zoom factor is at or beyond the specified limit, otherwise zoom is applied.
---	--

Properties

Optional.

Default

zoomReset on desktop computers; *none* on touch devices.

Example

singleclick=zoom

PageView.transition

[PageView. | <containerId>_pageView.]transition=*time* [, *easing*]

<i>time</i>	Specifies the time in seconds that the animation for a single zoom step action takes.
<i>easing</i>	<p>Creates an illusion of acceleration or deceleration which makes the transition appear more natural. You can set easing to one of the following:</p> <ul style="list-style-type: none">• 0 (auto)• 1 (linear)• 2 (quadratic)• 3 (cubic)• 4 (quartic)• 5 (quintic) <p>Auto mode always uses linear transition when elastic zoom is disabled (default). Otherwise, it fits one of the other easing functions based on the transition time. That is, the shorter the transition time the higher the easing function is used to accelerate the acceleration or deceleration effect.</p>

Properties

Optional.

Default

0.5,0

Example

transition=2,2

PageView.zoomstep

[PageView.|<containerId>_pageView.]zoomstep=step[,limit]

<i>step</i>	Configures the number of zoom in and zoom out actions that are required to increase or decrease the resolution by a factor of two. The resolution change for each zoom action is 2 ¹ per step. Set to 0 to zoom to full resolution with a single zoom action.
<i>limit</i>	Specifies the maximum zoom resolution, relative to the full resolution image. The default is 1.0, which does not allow zooming beyond full resolution.

Properties

Optional.

Default

1,1

Example

zoomstep=2,3

portraitFrames

portraitFrames=split|solid

split solid	Set to <code>split</code> to let the viewer display double-page spreads as a separate page when used on mobile device in portrait orientation. Set to <code>solid</code> to always display double-page spreads as solid image, irrespective to device orientation.
-------------	--

Properties

Optional.

Default

split

Example

portraitFrames=solid

Print.printquality

[Print. | <containerId>_print.]printquality=*size*

<i>size</i>	The maximum size of the image sent to print.
-------------	--

Properties

Optional.

Default

1000

Example

printquality=800

SearchPanel.align

[SearchPanel. | <containerId>_searchPanel.]align=left|center|right,top|center|bottom

left center right,top center bottom	<p>Specifies the internal horizontal alignment—or anchoring—of the thumbnails container within the component area.</p> <p>In SearchPanel, the internal thumbnail container is sized so that only a whole number of thumbnails are shown. As a result, there is some padding between the internal container and the external component bounds.</p> <p>This modifier specifies how the internal thumbnails container is positioned horizontally inside the component.</p>
-------------------------------------	---

Properties

Optional.

Default

center,top

Example

align=left,top

SearchPanel.fmt

[SearchPanel. | <containerId>_searchPanel.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha

jpg jpeg png png-alpha gif gif-alpha	Specifies the image format that the component uses for loading images from Image Server. It can be any format supported by Image Server and the client browser.
--------------------------------------	---

	If the specified format ends with <code>-alpha</code> , the component renders images as transparent content. For all other image formats the component treats images as opaque.
--	---

Properties

Optional.

Default

jpeg

Example

`fmt=png-alpha`

SearchPanel.iscommand

`[SearchPanel.<containerId>_searchPanel.]iscommand=isCommand`

<i>isCommand</i>	The Image Serving command string that is applied to all thumbnails. If specified in the URL all occurrences of <code>&</code> and <code>=</code> must be HTTP-encoded as <code>%26</code> and <code>%3D</code> , respectively.
------------------	--

Properties

Optional.

Default

None.

Example

When specified in the viewer URL.

`iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000`

When specified in the config data.

`iscommand=op_sharpen=1&op_colorize=0xff0000`

SearchPanel.maxloadradius

`[SearchPanel.<containerId>_searchPanel.]maxloadradius=-1|0|preloadnbr`

<code>-1 0 <i>preloadnbr</i></code>	<p>Specifies the component preload behavior.</p> <p>When set to <code>-1</code> all thumbnails are loaded simultaneously when the component is initialized or the asset changes.</p> <p>When set to <code>0</code> only visible thumbnails are loaded.</p> <p>Set <i>preloadnbr</i> to define how many invisible rows around the visible area are preloaded.</p>
-------------------------------------	--

Properties

Optional.

Default

1

Example

maxloadradius=0

SearchPanel.textpos

[SearchPanel.<containerId>_searchPanel.]textpos=bottom|top|left|right|none|tooltip

bottom top left right none tooltip	<p>Specifies where the label is drawn relative to the thumbnail image. That is, the label is centered at the specified location relative to the thumbnail.</p> <p>When tooltip is specified, the label text is displayed as a floating tooltip over the thumbnail image.</p> <p>When set to none, it turns label display off.</p>
------------------------------------	---

Properties

Optional.

Default

top

Example

textpos=bottom

searchServerUrl

searchServerUrl=searchServerUrl

searchServerUrl	<p>Search service root path. If no domain is specified, the domain from which the viewer is served is used.</p>
-----------------	---

Properties

Optional.

Default

/s7search/

Example

searchServerUrl=https://s7search1.scene7.com/s7search/

SocialShare.bearing

```
[SocialShare. |<containerId>_socialShare.]bearing= up|down|left|right|fit-vertical|fit-lateral
```

up down left right fit-vertical fit-lateral	<p>Specifies the direction of slide animation for the buttons container.</p> <p>When set to <code>up</code>, <code>down</code>, <code>left</code>, or <code>right</code>, the panel rolls out in a specified direction without an additional bounds check. This behavior can result in panel clipping by an outside container.</p> <p>When set to <code>fit-vertical</code>, the component first shifts the base panel position to the bottom of <code>SocialShare</code> and try to roll out the panel either from the bottom, right, or left, from such base location. With each attempt the component checks to see if panel is clipped by an outside container. If all attempts fail, the component tries to shift the base panel position to the top and repeat the roll out attempts from the top, right, and left direction.</p> <p>When set to <code>fit-lateral</code>, the component uses a similar logic as with <code>fit-vertical</code>, but instead shifts the base to the right first—trying right, down, and up roll out directions—and then shifts the base to the left, trying left, down, and up roll out directions.</p>
---	---

Properties

Optional.

Default

`fit-vertical`

Example

`bearing=left`

TableOfContents.bearing

```
[TableOfContents. |<containerId>_tableOfContents.]bearing=[fit-lateral|fit-vertical][,autoHideDelay]
```

fit-lateral fit-vertical	<p>Controls the direction of the drop-down panel appearance.</p> <p>When set to <code>fit-vertical</code>, the component first shifts the base panel position to the bottom of its button and tries to roll out the panel either to the right or to the left from the base location. With each attempt the component checks if the panel is clipped by an outside container. If all attempts fail, the component tries to shift the base panel position to the top and repeat roll out attempts in the right and left direction.</p> <p>When set to <code>fit-lateral</code>, the component uses a similar logic, but shifts the base to the right first, trying down and up roll out directions. Then, it shifts the base to the left, trying down and up roll out directions.</p>
<code>autoHideDelay</code>	<p>Sets the delay in seconds for the drop-down auto-hide timer which hides the panel when a user is idle.</p>

Properties

Optional.

Default

`fit-vertical,2`

Example

`bearing=fit-vertical,0.5`

TableOfContents.maxitems

`[TableOfContents.|<containerId>_tableOfContents.]maxitems=maxitems`

<i>maxitems</i>	<p>The maximum number of items in the drop-down table of contents.</p> <p>Additionally, you can decrease the number of visible items in the drop-down in case it becomes cropped by the outer container.</p> <p>When set to 0 that component shows as many items as possible given the container it is added to.</p>
-----------------	--

Properties

Optional.

Default

0

Example

`maxitems=10`

TableOfContents.showdefault

`[TableOfContents.|<containerId>_tableOfContents.]showdefault=0|1`

0 1	<p>When set to 1 the component populates the drop-down panel with elements for all pages, even for those that do not have label defined.</p> <p>When set to 0 only items with explicit labels show in the drop-down panel.</p>
-------	--

Properties

Optional.

Default

1

Example

showdefault=0

ThumbnailGridView.align

[ThumbnailGridView.<containerId>_gridView.]align=left|center|right

left center right	Specifies the internal horizontal alignment (anchoring) of the thumbnails container within the component area. In ThumbnailGridView, the internal thumbnail container is sized so that only a whole number of thumbnails is shown. As a result, there is some padding between the internal container and the external component bounds. This modifier specifies how the internal thumbnails container is positioned horizontally inside the component.
-------------------	--

Properties

Optional.

Default

left

Example

align=center

ThumbnailGridView.enabledragging

[ThumbnailGridView.<containerId>_gridView.]enabledragging=0|1[,overdragvalue]

0 1	Enables or disables the ability for a user to scroll the swatches with a mouse or by using touch gestures
overdragvalue	Functions within the 0–1 range. It is a % value for movement in the wrong direction of the actual speed. If it is set to 1, it moves with the mouse. If it is set to 0, it does not let you move in the wrong direction at all.

Properties

Optional.

Default

1,0.5

Example

enabledragging=0

ThumbnailGridView.fmt

```
[ThumbnailGridView. | <containerId>_gridView. ]fmt=jpg | jpeg | png | png-alpha | gif | gif-alpha
```

jpg jpeg png png-alpha gif gif-alpha	Specifies the image format that the component uses for loading images from Image Server. It can be any value that Image Server and the client browser supports. If the specified format ends with -alpha, the component renders images as transparent content. For all other image formats the component treats images as opaque.
--	---

Properties

Optional.

Default

jpeg

Example

fmt=png-alpha

ThumbnailGridView.maxloadradius

```
[ThumbnailGridView. | <containerId>_pageView. ]maxloadradius=-1 | 0 | preloadnbr
```

-1 0 <i>preloadnbr</i>	<p>Specifies the component preload behavior.</p> <p>When set to -1 the thumbnails are loaded simultaneously when the component is initialized or the asset changes.</p> <p>When set to 0 only the visible thumbnails are loaded.</p> <p>Set <i>preloadnbr</i> defines how many invisible rows/columns around the visible area are preloaded.</p>
----------------------------	--

Properties

Optional.

Default

1

Example

maxloadradius=0

ThumbnailGridView.scrollbar

```
[ThumbnailGridView. | <containerId>_gridView. ]scrollbar=0 | 1
```

0 1	Enables or disable the use of the scroll bar.
-------	---

Properties

Optional.

Default

1

Example

scrollbar=0

ThumbnailGridView.textpos

[ThumbnailGridView.<containerId>_gridView.]textpos=bottom|top|left|right|none|tooltip

bottom top left right none tooltip	<p>Specifies where the label is drawn relative to the thumbnail image. That is, the label is centered at the specified location relative to the thumbnail.</p> <p>When tooltip is specified, the label text is displayed as a floating tool tip over the thumbnail image. Set to none to turn off label.</p>
------------------------------------	--

Properties

Optional.

Default

bottom

Example

textpos=top

Javascript API reference for eCatalog Search Viewer

The main class of the eCatalog Search Viewer is eCatalogSearchViewer. It is declared in the s7viewers namespace. This JavaScript API covers constructor, methods, and callbacks of this particular class.

In all the following examples, <instance> stands for the actual name of the JavaScript viewer object that is instantiated from the s7viewers.eCatalogSearchViewer class.

dispose

JavaScript API reference for eCatalog Viewer.

dispose()

Disposes this viewer instance by releasing all resources used by the viewer logic and deleting all inner objects and components created by the viewer in runtime.

The web page code should also delete the viewer instance variable as well to completely remove the viewer from the web browser memory.

If the web page code has registered event listeners directly on Viewer SDK components used by the viewer—or stored external references to such components—such listeners must be explicitly unregistered by the web page code, and such external component references must be deleted prior to calling `dispose()`.

Do not access the Viewer API any more after `dispose()` is called.

Parameters

None.

Returns

None.

Example

```
<instance>.dispose()
```

eCatalogSearchViewer

JavaScript API reference for eCatalog SearchViewer.

```
eCatalogSearchViewer([config])
```

Constructor, creates a new eCatalog Search Viewer instance.

Parameters

<i>config</i>	<p>{Object} optional JSON configuration object, allows all the viewer settings to pass to the constructor and avoid calling individual setter methods. Contains the following properties:</p> <ul style="list-style-type: none">• <code>containerId</code> – {String} ID of the DOM container (normally a <code>DIV</code>) that the viewer is inserted into. It is not necessary to have the container element created by the time this method is called. However, the container must exist when <code>init()</code> is run. Required.• <code>params</code> – {Object} JSON object with viewer configuration parameters where the property name is either a viewer-specific configuration option or an SDK modifier, and the value of that property is a corresponding settings value. Required.• <code>handlers</code> – {Object} JSON object with viewer event callbacks, where the property name is the name of supported viewer event, and the property value is a JavaScript function reference to an appropriate callback. Optional. See Event callbacks for more information about viewer events.• <code>localizedTexts</code> – {Object} JSON object with localization data. Optional. See Localization of user interface elements for more information. <p>See also the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
---------------	--

Returns

None.

Example

```
var eCatalogSearchViewer = new s7viewers.eCatalogSearchViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Viewers/Pluralist",
    "serverurl": "http://s7dl.scene7.com/is/image/",
    "searchserverurl": "http://s7search1.scene7.com/s7search/"
  },
  "handlers": {
    "initComplete": function() {
      console.log("init complete");
    }
  },
  "localizedTexts": {
    "en": {
      "CloseButton.TOOLTIP": "Close"
    },
    "fr": {
      "CloseButton.TOOLTIP": "Fermer"
    }
  },
  defaultLocale: "en"
});
```

getComponent

JavaScript API reference for eCatalog Viewer

`getComponent(componentId)`

Returns a reference to the Viewer SDK component that is used by the viewer. The web page can use this method to extend or customize the behavior of the out-of-box viewer. Call this method only after the `initComplete` viewer callback has run, otherwise the component may not be created yet by the viewer logic.

Parameters

componentID - {String} an ID of the Viewer SDK component used by the viewer. This viewer supports the following component IDs:

Component ID	Viewer SDK component class name
parameterManager	s7sdk.ParameterManager
container	s7sdk.common.Container
mediaSet	s7sdk.set.MediaSet
pageView	s7sdk.set.PageView
primaryControls	s7sdk.common.ControlBar
secondaryControls	s7sdk.common.ControlBar

Component ID	Viewer SDK component class name
gridView	s7sdk.set.ThumbnailGridView
tableOfContents	s7sdk.set.TableOfContents
infoPanelPopup	s7sdk.info.InfoPanelPopup
imageMapEffect	s7sdk.image.ImageMapEffect
leftButton	s7sdk.common.PanLeftButton
rightButton	s7sdk.common.PanRightButton
zoomInButton	s7sdk.common.ZoomInButton
zoomOutButton	s7sdk.common.ZoomOutButton
zoomResetButton	s7sdk.common.ZoomResetButton
secondaryZoomResetButton	s7sdk.common.ZoomResetButton
thumbnailPageButton	s7sdk.common.ThumbnailPageButton
fullScreenButton	s7sdk.common.FullScreenButton
toolBarLeftButton	s7sdk.common.PanLeftButton
toolBarRightButton	s7sdk.common.PanRightButton
firstPageButton	s7sdk.common.PanLeftButton
secondaryFirstPageButton	s7sdk.common.PanLeftButton
lastPageButton	s7sdk.common.PanRightButton
secondaryLastPageButton	s7sdk.common.PanRightButton
closeButton	s7sdk.common.CloseButton
socialShare	s7sdk.share.SocialShare
twitterShare	s7sdk.share.TwitterShare

Component ID	Viewer SDK component class name
facebookShare	s7sdk.share.FacebookShare
linkShare	s7sdk.share.LinkShare
emailShare	s7sdk.share.EmailShare
embedShare	s7sdk.share.EmbedShare
print	s7sdk.share.Print
download	s7sdk.common.Download
favoritesEffect	s7sdk.favorites.FavoritesEffect
favoritesView	s7sdk.favorites.FavoritesView
favoritesMenu	s7sdk.favorites.FavoritesMenu
addFavoriteButton	s7sdk.favorites.AddFavoriteButton
removeFavoriteButton	s7sdk.favorites.RemoveFavoriteButton
viewAllFavoriteButton	s7sdk.favorites.ViewAllFavoriteButton
searchButton	s7sdk.common.SearchButton
searchPanel	s7sdk.search.SearchPanel
searchManager	s7sdk.search.SearchManager
searchEffect	s7sdk.search.SearchEffect

When working with SDK APIs it is important to use correct fully qualified SDK namespace as described in [Viewer SDK namespace](#).

See the *Viewer SDK API* documentation for more information about a particular component.

Returns

{Object} a reference to Viewer SDK component. The method returns null if the `componentId` is not a supported viewer component or if the component was not yet created by the viewer logic.

Example

```
<instance>.setHandlers({  
  "initComplete":function() {
```

```
var pageView = <instance>.getComponent("pageView");
}
})
```

init

JavaScript API reference for eCatalog Viewer.

```
init()
```

Starts the initialization of the eCatalog Viewer. By this time container DOM element must be created so that the viewer code can find it by its ID.

If the container element is not a part of the web page layout just yet (for example, it may be hidden using `display:none` style assigned to it), the viewer suspends its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load automatically resumes.

Only call this method once during the viewer life cycle; subsequent calls are ignored.

Parameters

None.

Returns

{Object} A reference to viewer instance.

Example

```
<instance>.init()
```

setAsset

JavaScript API reference for Video Viewer.

```
setAsset(asset)
```

<i>asset</i>	<p>{String} new asset id or explicit image set with optional Image Serving modifiers appended after ?.</p> <p>Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.</p>
--------------	---

Sets a new asset. You can call this parameter at any time, either before or after `init()`. If it is called after `init()`, the viewer swaps the asset at runtime.

See also [init](#).

Returns

None.

Example

Single reference to an image set that is defined in a catalog:

```
<instance>.setAsset("Viewers/Pluralist")
```

Explicit image set, with pre-combined pages:

```
<instance>.setAsset("Scene7SharedAssets/Backpack_B,Scene7SharedAssets/Backpack_C,Scene7SharedAssets/Backpack_H,Scene7SharedAssets/Backpack_J")
```

Explicit image set, with individual page images:

```
<instance>.setAsset("Scene7SharedAssets/AdobeScene7OverviewUS-1,Scene7SharedAssets/AdobeScene7OverviewUS-2,AdobeScene7OverviewUS-3,Scene7SharedAssets/AdobeScene7OverviewUS-4")
```

Sharpening modifier added to all images in the set:

```
<instance>.setAsset("Viewers/Pluralist?op_sharpen=1")
```

setContainerId

JavaScript API reference for eCatalog Viewer.

```
setContainerId(containerId)
```

Sets the ID of the DOM container (normally a DIV) into which the viewer is inserted. It is not necessary to have the container element created by the time this method is called. However, the container must exist when `init()` is run. It must be called before `init()`.

This method is optional if the viewer configuration information is passed with `config` JSON object to the constructor.

<i>containerId</i>	{string} ID of container.
--------------------	---------------------------

Returns

None.

Example

```
<instance>.setContainerId("s7viewer");
```

setHandlers

JavaScript API reference for eCatalog Viewer.

```
setHandlers(handlers)
```

Specifies zero or more callback handlers. A call to this method fully overwrites event handlers that were previously assigned for that viewer instance. Must be called before `init()`.

Parameter

<i>handlers</i>	<p>{Object} JSON object with viewer event callbacks, where the property name is the name of the supported viewer event and the property value is a JavaScript function reference to an appropriate callback.</p> <p>See Event callbacks for more information about viewer events.</p>
-----------------	---

Returns

None.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    console.log("init complete");
  }
})
```

setLocalizedTexts

JavaScript API reference for Video Viewer.

setLocalizedTexts(*localizationInfo*)

<i>localizationInfo</i>	<p>{Object} JSON object with localization data.</p> <p>See Localization of user interface elements for more information.</p> <p>See also the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
-------------------------	--

Sets localization SYMBOL values for one or more locales. This parameter must be called before `init()`.

See also [init](#).

Returns

None.

Example

```
<instance>.setLocalizedTexts({"en":{"CloseButton.TOOLTIP":"Close"},"fr":{"CloseButton.TOOLTIP":"Femer"},defaultLocale:"en"})
```

setParam

JavaScript API reference for eCatalog Viewer.

setParam(*name*, *value*)

Sets the viewer parameter to a specified value. The parameter is either a viewer-specific configuration option or a software development kit modifier. This parameter is called before `init()`.

This method is optional if the viewer configuration information is passed with `config` JSON object to the constructor.

See also [init](#).

<i>name</i>	{string} name of parameter.
<i>value</i>	{string} value of parameter. The value cannot be percent-encoded.

Returns

None.

Example

```
<instance>.setParam("style", "customStyle.css")
```

setParams

JavaScript API reference for eCatalog Viewer.

`setParams(params)`

Sets one or more parameters to a given value. The method argument syntax is identical to a URL query string. That is, it represents name=value pairs separated with &. Just as in a query string, the names and values are percent-encoded using UTF8. Before you call `init()`, this parameter must be called.

This method is optional if the viewer configuration information is passed with `config` JSON object to the constructor.

See also [init](#).

<code>params</code>	{string} name=value parameter pairs separated with &.
---------------------	---

Returns

None.

Example

```
<instance>.setParams("PageView.zoomstep=2,3&PageView.iscommand=op_sharpen%3d1")
```

Event callbacks

The viewer supports JavaScript event callbacks that the web page uses to track the viewer initialization process or runtime behavior.

Callback handlers are assigned by passing event names and corresponding handler functions with the `handlers` property to `config` JSON object in the viewer's constructor. Alternatively, it is possible to use `setHandlers()` API method.

Supported viewer events include the following:

- `initComplete` – triggers when viewer initialization is complete and all internal components are created, so that it is possible to use `getComponent()` API. The callback handler does not take any arguments.
- `trackEvent` – triggers each time an event occurs inside the viewer which may be handled by an event tracking system, such as Adobe Analytics. The callback handler takes the following arguments:
 - `objID` {String} not currently used.
 - `compClass` {String} not currently used.
 - `instName` {String} an instance name of the Viewer SDK component that triggered the event.
 - `timeStamp` {Number} event time stamp.
 - `eventInfo` {String} event payload.

See also `and` [setHandlers](#).

Customizing eCatalog Search Viewer

All visual customization and most behavior customization for the eCatalog Search Viewer is done by creating a custom CSS.

The suggested workflow is to take the default CSS file for the appropriate viewer, copy it to a different location, customize it, and specify the location of the customized file in the `style=` command.

Default CSS files can be found at the following location:

```
<s7_viewers_root>/html5/eCatalogSearchViewer_dark.css
```

The custom CSS file must contain the same class declarations as the default one. If a class declaration is omitted, the viewer does not function properly because it does not provide built-in default styles for the user interface elements.

An alternative way to provide custom CSS rules is to use embedded styles directly on the web page or in one of the linked external CSS rules.

When creating custom CSS keep in mind that the viewer assigns `.s7ecatalogsearchviewer` class to its container DOM element. If you are using external CSS file passed with the `style=` command, use `.s7ecatalogsearchviewer` class as parent class in descendant selector for your CSS rules. If you are doing embedded styles on the web page, additionally qualify this selector with an ID of the container DOM element as follows:

```
#<containerId>.s7ecatalogsearchviewer
```

Building responsive designed CSS

It is possible to target different devices and embedding sizes in CSS to make your content display differently, depending on a user's device or a particular web page layout. This includes, but is not limited to, different web page layouts, user interface element sizes, and artwork resolution.

The viewer supports two methods for creating responsive designed CSS: CSS markers and standard CSS media queries. You can use these methods independently or together.

CSS markers

To assist in creating responsive designed CSS, the viewer supports CSS markers which special CSS classes dynamically assigned to the top-level viewer container element based on the run-time viewer size and the input type used on the current device.

The first group of CSS markers includes `.s7size_large`, `.s7size_medium`, and `.s7size_small` classes. They are applied based on the runtime area of the viewer container. That is, if the viewer area is equal to or bigger than the size of a common desktop monitor `.s7size_large` is used; if the area is close in size to a common tablet device `.s7size_medium` is assigned. For areas similar to mobile phone screens `.s7size_small` is set. The primary purpose of these CSS markers is to create different user interface layouts for different screens and viewer sizes.

The second group of CSS Markers includes `.s7mouseinput` and `.s7touchinput`. `.s7touchinput` is set if the current device has touch input capabilities; otherwise, `.s7mouseinput` is used. These markers are intended to create user interface input elements with different screen sizes for different input types, because normally touch input requires larger elements. In case the device has both mouse input and touch capabilities, `.s7touchinput` is set and the viewer renders a touch-friendly user interface.

The following sample CSS sets the zoom in button size to 28 x 28 pixels on systems with mouse input, and 56 x 56 pixels on touch devices. In addition, it hides the button completely if the viewer size becomes really small:

```
.s7ecatalogsearchviewer.s7mouseinput .s7zoominbutton {
    width:28px;
    height:28px;
}
.s7ecatalogsearchviewer.s7touchinput .s7zoominbutton {
    width:56px;
    height:56px;
}
.s7ecatalogsearchviewer.s7size_small .s7zoominbutton {
    visibility:hidden;
}
```

To target devices with a different pixel density, use CSS media queries. The following media query block would contain CSS that is specific to high density screens:

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5)
{
}
```

Using CSS markers is the most flexible way of building responsive designed CSS as it allows you to target not only device screen size but actual viewer size, which may be useful for responsive designed page layouts.

Use the default viewer CSS file as an example of a CSS markers approach.

CSS media queries

Device sensing can also be done using pure CSS media queries. Everything enclosed within a given media query block is applied only when it is run on a corresponding device.

When applied to Mobile Viewers, use four CSS media queries, defined in your CSS in the following order:

1. Contains only rules specific for all touch devices.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px)
{
}
```

2. Contains only rules specific for tablets with high resolution screens.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px) and (-webkit-min-device-pixel-ratio:1.5),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px) and (-webkit-min-device-pixel-ratio:1.5)
{
}
```

3. Contains only rules specific for all mobile phones.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in)
{
}
```

4. Contains only rules specific for mobile phones with high resolution screens.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in) and
(-webkit-min-device-pixel-ratio: 1.5),
only screen and (device-width:720px) and (device-height:1280px) and
(-webkit-device-pixel-ratio: 2),
only screen and (device-width:1280px) and (device-height:720px) and
(-webkit-device-pixel-ratio: 2)
{
}
```

Using a media queries approach, you should organize CSS with device sensing as follows:

- First, the desktop-specific section defines all properties that are either desktop-specific or common to all screens.
- And second, the four media queries go in the order defined above and provide CSS rules that are specific for the corresponding device type.

There is no need to duplicate the entire viewer CSS in each media query. Only properties that are specific to given devices are redefined inside a media query.

CSS Sprites

Many viewer user interface elements are styled using bitmap artwork and have more than one distinct visual state. A good example is a button that normally has at least three different states: "up", "over", and "down". Each state requires its own bitmap artwork assigned.

With a classic approach to styling, the CSS would have a separate reference to individual image file on the server for each state of the user interface element. The following is a sample CSS for styling a zoom-in button:

```
.s7ecatalogsearchviewer.s7mouseinput .s7zoominbutton[state='up'] {
background-image:url(images/v2/ZoomInButton_dark_up.png);
}
.s7ecatalogsearchviewer.s7mouseinput .s7zoominbutton[state='over'] {
background-image:url(images/v2/ZoomInButton_dark_over.png);
}
.s7ecatalogsearchviewer.s7mouseinput .s7zoominbutton[state='down'] {
background-image:url(images/v2/ZoomInButton_dark_down.png);
}
.s7ecatalogsearchviewer.s7mouseinput .s7zoominbutton[state='disabled'] {
background-image:url(images/v2/ZoomInButton_dark_disabled.png);
}
```

The drawback to this approach is that the end user experiences flickering or delayed user interface response when the element is interacted with for the first time. This action occurs because the image artwork for the new element state is not yet downloaded. Also, this approach may have a slight negative impact on performance because of an increase in the number of HTTP calls to the server.

CSS sprites is a different approach where image artwork for all element states is combined into a single PNG file called a "sprite". Such "sprite" has all visual states for the given element positioned one after another. When styling a user interface element with sprites the same sprite image is referenced for all different states in the CSS. Also, the `background-position` property is used for each state to specify which part of the "sprite" image is used. You can structure a "sprite" image in any suitable way. Viewers normally have it vertically stacked. Below is a "sprite"-based example of styling the same zoom-in button from above:

```
.s7ecatalogsearchviewer .s7zoominbutton[state] {
background-image: url(images/v2/ZoomInButton_dark_sprite.png);
}
.s7ecatalogsearchviewer.s7mouseinput .s7zoominbutton[state='up'] {
background-position: -84px -560px;
}
.s7ecatalogsearchviewer.s7mouseinput .s7zoominbutton[state='over'] {
background-position: -56px -560px;
}
.s7ecatalogsearchviewer.s7mouseinput .s7zoominbutton[state='down'] {
background-position: -28px -560px;
}
.s7ecatalogsearchviewer.s7mouseinput .s7zoominbutton[state='disabled'] {
background-position: -0px -560px;
}
```

General styling notes and advice

- When customizing the viewer user interface with CSS the use of the `!IMPORTANT` rule is not supported to style viewer elements. In particular, `!IMPORTANT` rule should not be used to override any default or run-time styling provided by the viewer or Viewer SDK. The reason is that it may affect the behavior of proper components. Instead, you should use CSS selectors with the proper specificity to set CSS properties that are documented in this reference guide.
- All paths to external assets within CSS are resolved against the CSS location, not the viewer HTML page location. Be aware of this rule when you copy the default CSS to a different location. Either copy the default assets as well or update paths within the custom CSS.
- The preferred format for bitmap artwork is PNG.

- Bitmap artwork is assigned to user interface elements using the `background-image` property.
- The `width` and `height` properties of a user interface element define its logical size. The size of the bitmap passed to `background-image` does not affect logical size.
- To use the high pixel density of high-resolution screens like Retina, specify bitmap artwork twice as large as the logical user interface element size. Then, apply the `-webkit-background-size:contain` property to scale the background down to the logical user interface element size.
- To remove a button from the user interface, add `display:none` to its CSS class.
- You can use various formats for color value that CSS supports. If you need transparency, use the format `rgba(R,G,B,A)`. Otherwise, you can use the format `#RRGGBB`.

Common User Interface Elements

The following is user interface elements reference documentation that applies to eCatalog Search Viewer:

Add Favorite button

The position of the Add Favorite button is fully managed by the Favorites menu.

The appearance of the Add Favorite button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7addfavoritebutton
```

CSS properties of the Add Favorite button

<code>background-image</code>	The image that is displayed for a given button state.
<code>background-position</code>	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
<code>width</code>	Width of the button.
<code>height</code>	Height of the button.



Note: This button supports both the `state` and `selected` attribute selectors, which can be used to apply different skins to different button states. In particular, `selected='true'` corresponds to the state when a user can add a new Favorite icon by clicking or tapping. `selected='false'` corresponds to the normal operation mode when a user can zoom, pan, and swap pages.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Add Favorite button that is 28 x 28 pixels, and displays a different image for each of the four different button states when selected or not selected.

```
.s7ecatalogsearchviewer .s7addfavoritebutton {
  width:28px;
  height:28px;
}
.s7ecatalogsearchviewer .s7addfavoritebutton[selected='false'][state='up'] {
  background-image:url(images/v2/AddFavoriteButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7addfavoritebutton[selected='false'][state='over'] {
```

```
background-image:url(images/v2/AddFavoriteButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7addfavoritebutton[selected='false'][state='down'] {
background-image:url(images/v2/AddFavoriteButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7addfavoritebutton[selected='false'][state='disabled'] {
background-image:url(images/v2/AddFavoriteButton_dark_disabled.png);
}
.s7ecatalogsearchviewer .s7addfavoritebutton[selected='true'][state='up'] {
background-image:url(images/v2/AddFavoriteButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7addfavoritebutton[selected='true'][state='over'] {
background-image:url(images/v2/AddFavoriteButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7addfavoritebutton[selected='true'][state='down'] {
background-image:url(images/v2/AddFavoriteButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7addfavoritebutton[selected='true'][state='disabled'] {
background-image:url(images/v2/AddFavoriteButton_dark_disabled.png);
}
}
```

Close button

Clicking or tapping this button closes the containing web page. This button only appears if the `closebutton` parameter is set to 1. This button is not available on desktop systems. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7closebutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.
left	Position from the left border of the main control bar, including padding.
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which you can use to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a close button that is 56 x 56 pixels, positioned 4 pixels from the top and right edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7closebutton {
top:4px;
right:4px;
width:56px;
height:56px;
}
.s7ecatalogsearchviewer .s7closebutton [state='up'] {
background-image:url(images/v2/CloseButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7closebutton [state='over'] {
background-image:url(images/v2/CloseButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7closebutton [state='down'] {
background-image:url(images/v2/CloseButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7closebutton [state='disabled'] {
background-image:url(images/v2/CloseButton_dark_disabled.png);
}
```

Download

The appearance of the Download button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7download
```

CSS properties of the Download button

margin-top	The offset from the top of the control bar.
margin-left	The distance to the next button on the left, or the left side of the control bar if this is the first button in a row.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Download button that is 28 x 28 pixels and displays a different image for each of the four different button states:

```
.s7ecatalogsearchviewer .s7download {
margin-top: 4px;
margin-left: 10px;
width:28px;
height:28px;
```

```

}
.s7ecatalogsearchviewer .s7download[state='up'] {
background-image:url(images/v2/Download_dark_up.png);
}
.s7ecatalogsearchviewer .s7download[state='over'] {
background-image:url(images/v2/Download_dark_over.png);
}
.s7ecatalogsearchviewer .s7download[state='down'] {
background-image:url(images/v2/Download_dark_down.png);
}
.s7ecatalogsearchviewer .s7download[state='disabled'] {
background-image:url(images/v2/Download_dark_disabled.png);
}

```

Email share

Email share tool consists of a button added to the Social share panel and the modal dialog box which displays when the tool is activated. The position of the button is fully managed by the Social share tool.

The appearance of the email share button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emailshare
```

CSS properties of the email share tool

width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a email share button that is 28 x 28 pixels, and that displays a different image for each of the four different button states.

```

.s7ecatalogsearchviewer .s7emailshare {
width:28px;
height:28px;
}
.s7ecatalogsearchviewer .s7emailshare[state='up'] {
background-image:url(images/v2/EmailShare_dark_up.png);
}
.s7ecatalogsearchviewer .s7emailshare[state='over'] {
background-image:url(images/v2/EmailShare_dark_over.png);
}
.s7ecatalogsearchviewer .s7emailshare[state='down'] {
background-image:url(images/v2/EmailShare_dark_down.png);
}
.s7ecatalogsearchviewer .s7emailshare[state='disabled'] {
background-image:url(images/v2/EmailShare_dark_disabled.png);
}

```

The background overlay which covers web page when the dialog is active is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7backoverlay
```

CSS properties of the back overlay

opacity	Background overlay opacity.
background-color	Background overlay color.

Example – to set up background overlay to be gray with 70% opacity:

```
.s7ecatalogsearchviewer .s7emaildialog .s7backoverlay {
  opacity:0.7;
  background-color:#222222;
}
```

By default the modal dialog is displayed centered in the screen on desktop systems and takes the whole web page area on touch devices. In all cases, the positioning and sizing of the dialog box is managed by the component. The dialog is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialog
```

CSS properties of the dialog box

border-radius	Dialog box border radius (in case the dialog box does not take the entire browser window);
background-color	Dialog box background color;
width	Should be either unset, or set to 100%, in which case the dialog takes the whole browser window (this mode is preferred on touch devices);
height	Should be either unset, or set to 100%, in which case the dialog takes the whole browser window (this mode is preferred on touch devices).

Example – to set up dialog to use the entire browser window and have a white background on touch devices:

```
.s7ecatalogsearchviewer .s7touchinput .s7emaildialog .s7dialog {
  width:100%;
  height:100%;
  background-color: #ffffff;
}
```

The dialog box header consists of an icon, a title text and a close button. The header container is controlled with the following CSS class selector

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogheader
```

CSS properties of the dialog box header

padding	Inner padding for header content.
---------	-----------------------------------

The icon and the title text are wrapped into an additional container controlled with

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogheader .s7dialogline
```


CSS properties of the dialog line

padding	Inner padding for the header icon and title.
---------	--

Header icon is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogheadericon
```

CSS properties of the dialog box header icon

width	Icon width.
height	Icon height.
background-image	Icon image.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .

Header title is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogheadertext
```

CSS properties of the dialog box header text

font-weight	Font weight.
font-size	Font height.
font-family	Font family.
padding	Internal text padding.

Close button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7closebutton
```

CSS properties of the close button

top	Vertical button position relative to header container.
right	Horizontal button position relative to header container.
width	Button width.
height	Button height.
padding	Inner padding of button.

background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The Close button tool tip and the dialog box title can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up dialog header with padding, 24 x 17 pixels icon, bold 16 point title, and a 28 x 28 pixel Close button positioned two pixels from the top and two pixels from the right of dialog box container:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogheader {
  padding: 10px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogheader .s7dialogline {
  padding: 10px 10px 2px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogheadericon {
  background-image: url("images/sdk/dlgemail_cap.png");
  height: 17px;
  width: 24px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogheadertext {
  font-size: 16pt;
  font-weight: bold;
  padding-left: 16px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7closebutton {
  top: 2px;
  right: 2px;
  padding: 8px;
  width: 28px;
  height: 28px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7closebutton[state='up'] {
  background-image: url(images/sdk/close_up.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7closebutton[state='over'] {
  background-image: url(images/sdk/close_over.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7closebutton[state='down'] {
  background-image: url(images/sdk/close_down.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7closebutton[state='disabled'] {
  background-image: url(images/sdk/close_disabled.png);
}
```

Dialog footer consists of Cancel and Send email buttons. The footer container is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogfooter
```

CSS properties of the dialog box footer

border	Border that you can use to visually separate the footer from the rest of the dialog box.
--------	--

The footer has an inner container which keeps both buttons. It is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogbuttoncontainer
```

CSS properties of the dialog box button container

padding	Inner padding between the footer and the buttons.
---------	---

Cancel button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogcancelbutton
```

CSS properties of the dialog box cancel button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

Send email button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogactionbutton
```

CSS properties of the dialog box action button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

In addition, both buttons share the same common CSS class which can contain CSS settings that are the same for other dialog box buttons:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogfooter .s7button
```

CSS properties of the button

font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.
line-height	Text height inside the button. Affects vertical alignment.

box-shadow	Drop shadow.
margin-right	Right button margin.

This buttons tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box footer with 64 x 34 Cancel button and a 82 x 34 send email button, with the text color and background color different for each button state:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogfooter {
  border-top: 1px solid #909090;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogbuttoncontainer {
  padding-bottom: 6px;
  padding-top: 10px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogfooter .s7button {
  box-shadow: 1px 1px 1px #999999;
  color: #FFFFFF;
  font-size: 9pt;
  font-weight: bold;
  line-height: 34px;
  margin-right: 10px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogcancelbutton {
  width: 64px;
  height: 34px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogcancelbutton[state='up'] {
  background-color: #666666;
  color: #dddddd;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogcancelbutton[state='down'] {
  background-color: #555555;
  color: #ffffff;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogcancelbutton[state='over'] {
  background-color: #555555;
  color: #ffffff;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogcancelbutton[state='disabled'] {
  background-color: #b2b2b2;
  color: #dddddd;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogactionbutton {
  width: 82px;
  height: 34px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogactionbutton[state='up'] {
  background-color: #333333;
  color: #dddddd;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogactionbutton[state='down'] {
  background-color: #222222;
  color: #cccccc;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogactionbutton[state='over'] {
  background-color: #222222;
  color: #cccccc;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogactionbutton[state='disabled'] {
  background-color: #b2b2b2;
  color: #dddddd;
}
```

The main dialog area (between the header and the footer) contains scrollable dialog content and scroll panel on the right. In all cases, the component manages the width of this area, it is not possible to set it in CSS. Main dialog area is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogviewarea
```

CSS properties of the dialog box viewing area

height	The height of the main dialog box area. It should be specified only when the dialog box works in desktop mode. It is not applicable when the dialog box is sized to occupy the entire browser window.
background-color	The background color of the main dialog box area.
margin	Outer margin.



Note: The main dialog box area supports the optional state attribute selector. It is set to `sendsuccess` when the email form is submitted and the dialog box shows a confirmation message. As long as the confirmation message is small, this attribute selector can be used to reduce the dialog box height when such confirmation message is displayed.

Example – to set up the main dialog box area to be 300 pixels height initially and 100 pixels height when the confirmation message is shown, have a ten pixel margin, and use a white background:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogviewarea {
  background-color:#ffffff;
  margin:10px;
  height:300px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogviewarea[state='sendsuccess'] {
  height:100px;
}
```

All form content (like labels and input fields) resides inside a container controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogbody
```

If the height of this container appears to be bigger than the main dialog box area, a vertical scroll is enabled automatically by the component.

CSS properties of the dialog box body

padding	Inner padding.
---------	----------------

Example – to set up form content to have ten pixel padding:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogbody {
  padding: 10px;
}
```

Dialog box form is filled on line-by-line basis, where each line carries a part of the form content (like a label and a text input field). Single form line is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogbody .s7dialogline
```

CSS properties of the dialog box line

padding	Inner line padding.
---------	---------------------

Example – to set up a dialog box form to have ten pixel padding for each line:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogbody .s7dialogline {
  padding: 10px;
}
```

All static labels in the dialog box form are controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialoglabel
```

This class is not suitable for controlling labels size or position because you can apply it to texts in various places of the form user interface.

CSS properties of the dialog box label.

font-weight	Label font weight.
font-size	Label font size.
font-family	Label font family.
color	Label text color.

Dialog box labels can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up all labels to be gray, bold, with a nine pixel font:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialoglabel {
  color: #666666;
  font-size: 9pt;
  font-weight: bold;
}
```

All static labels that are displayed to the left of the form input fields are controlled with:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialoginputlabel
```

CSS properties of the dialog box input label

width	The width of the static label.
text-align	The horizontal text alignment.
margin	Static label margin.
padding	Static label padding.

Example – to set up input field labels to be 50 pixels width, right-aligned, have ten pixels of padding, and a ten pixel margin on the right:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialoginputlabel {
  margin-right: 10px;
  padding: 10px;
}
```

```

    text-align: right;
    width: 50px;
}

```

Each form input field is wrapped into the container which lets you apply a custom border around the input field. It is controlled with the following CSS class selector::

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

border	Border around the input field container.
padding	Inner padding.



Note: Input field container supports optional *state* attribute selector. It is set to *verifyerror* when the user makes a mistake in input data format and inline validation fails. This attribute selector can be used to highlight incorrect user input in the form.

Most input fields that spread from the label on the left up to the right edge of the dialog box body (which includes From field and Message field) are controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialoginputwide
```

CSS properties of the dialog box input wide field

width	Input field width.
-------	--------------------

The To field is narrower because it allocates space for the Remove email button on the right. It is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialoginputshort
```

CSS properties of the dialog box input short field

width	Input field width.
-------	--------------------

Example – to set up a form to have a one pixel grey border with nine pixels of padding around all input fields; to have the same border in red color for fields which fail validation, to have 250 pixels wide To field, and the rest of the input fields 300 pixels wide:

```

.s7ecatalogsearchviewer .s7emaildialog .s7dialoginputcontainer {
    border: 1px solid #CCCCCC;
    padding: 9px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialoginputcontainer[state="verifyerror"] {
    border: 1px solid #FF0000;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialoginputshort {
    width: 250px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialoginputwide {
    width: 300px;
}

```

Email message input field is additionally controlled with:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogmessage
```

This class lets you set specific properties for the underlying TEXTAREA element.

CSS properties of the dialog box message

height	Message height.
word-wrap	Word wrapping style.

Example – to set up an email message to be 50 pixels high and use break-word word wrapping:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogmessage {
    height: 50px;
    word-wrap: break-word;
}
```

Add Another Email Address button lets a user add more than one addressee in email form. It is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogaddemailbutton
```

CSS properties of the dialog box add email address button

height	Button height.
color	Button text color for each state.
background-image	Button image for each state.
background-position	Button image position inside the button area.
font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.
line-height	Text height inside the button. Affects the vertical alignment.
text-align	Horizontal text alignment.
padding	Inner padding.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up "Add Another Email Address" button to be 25 pixels high, use 12 point bold font with right alignment, and a different text color and image for each state:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogaddemailbutton {
  text-align:right;
  font-size:12pt;
  font-weight:bold;
  background-position:left center;
  line-height:25px;
  padding-left:30px;
  height:25px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogaddemailbutton[state='up'] {
  color:#666666;
  background-image:url(images/sdk/dlgaddplus_up.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogaddemailbutton[state='down'] {
  color:#000000;
  background-image:url(images/sdk/dlgaddplus_over.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogaddemailbutton[state='over'] {
  color:#000000;
  text-decoration:underline;
  background-image:url(images/sdk/dlgaddplus_over.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogaddemailbutton[state='disabled'] {
  color:#666666;
  background-image:url(images/sdk/dlgaddplus_up.png);
}
```

Remove button lets a user remove extra addressees from the email form. It is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogremoveemailbutton
```

CSS properties of the dialog box remove email button

width	Button width.
height	Button height.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Remove button to be 25 x 25 pixels and use a different image for each state:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogremoveemailbutton {
  width:25px;
  height:25px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogremoveemailbutton[state='up'] {
  background-image:url(images/sdk/dlgremove_up.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogremoveemailbutton[state='over'] {
  background-image:url(images/sdk/dlgremove_over.png);
}
```

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogremoveemailbutton[state='down'] {
  background-image:url(images/sdk/dlgremove_over.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogremoveemailbutton[state='disabled'] {
  background-image:url(images/sdk/dlgremove_up.png);
}
```

The content being shared is displayed in the bottom of the dialog box body and includes a thumbnail, title, origin URL, and description. It is wrapped into a container that is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogbody .s7dialogcontent
```

CSS properties of the dialog box content

border	Container border.
padding	Inner padding.

Example – to set up a bottom container to have a one pixel dotted border and no padding:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogbody .s7dialogcontent {
  border: 1px dotted #A0A0A0;
  padding: 0;
}
```

Thumbnail image is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogthumbnail
```

The background-image property is set by the component logic.

CSS properties of the dialog box thumbnail image

width	Thumbnail width.
height	Thumbnail height.
vertical-align	Vertical alignment thumbnail.
padding	Inner padding.

Example – to set up thumbnail to be 90 x 60 pixels, and top-aligned with ten pixels of padding:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogthumbnail {
  height: 60px;
  padding: 10px;
  vertical-align: top;
  width: 90px;
}
```

Content title, origin, and description are further grouped into a panel to the right of the content thumbnail. It is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialoginfopanel
```

CSS properties of the dialog box information panel

width	Panel width.
-------	--------------

Example – to set up a content information panel to be 300 pixels wide:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialoginfopanel {
  width: 300px;
}
```

Content title is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogtitle
```

CSS properties of the dialog box title

margin	Outer margin.
font-weight	Font weight.
font-size	Font size.
font-family	Font family.

Example – to set up a content title to use bold font and have a ten pixel margin:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogtitle {
  font-weight: bold;
  margin: 10px;
}
```

Content origin is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogorigin
```

CSS properties of the dialog box content origin

margin	Outer margin.
font-weight	Font weight.
font-size	Font size.
font-family	Font family.

Example – to set up content origin to have a ten pixel margin:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogorigin {
  margin: 10px;
}
```

Content description is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogdescription
```

CSS properties of the dialog box content description

margin	Outer margin.
font-weight	Font weight.

font-size	Font size.
font-family	Font family.

Example – to set up a content description to have a ten pixel margin and use a nine point font:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogdescription {
  font-size: 9pt;
  margin: 10px;
}
```

When a user enters incorrect input data and inline validation fails, or when the dialog box needs to render an error or a confirmation message when the form is submitted, a message is displayed in the top of the dialog box body. It is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogerrormessage
```

CSS properties of the dialog box error message

background-image	Error icon. Default is an exclamation mark.
background-position	Error icon position inside the message area.
color	Message text color.
font-weight	Font weight.
font-size	Font size.
font-family	Font family.
line-height	Text height inside the message. Affects vertical alignment.
padding	Inner padding.



Note: This message supports the *state* attribute selector with the following possible values: *verifyerror*, *senderror*, and *sendsuccess*. *verifyerror* is set when a message is displayed due to an inline input validation failure; *senderror* is set when a backend email service reports an error; *sendsuccess* is set when email is sent successfully. This way it is possible to style the message differently depending on the dialog box state.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a message to use a ten point bold font, have 25 pixels line height, 20 pixels padding on the left, use an exclamation mark icon, red text in case of an error, and no icon and green text in case of success:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogerrormessage[state="verifyerror"] {
  background-image: url("images/sdk/dlgerrimg.png");
  color: #FF0000;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogerrormessage[state="senderror"] {
  background-image: url("images/sdk/dlgerrimg.png");
  color: #FF0000;
}
```

```
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogerrormessage[state="sendsuccess"] {
  background-image: none;
  color: #00B200;
}
.s7ecatalogsearchviewer .s7emaildialog .s7dialogerrormessage {
  background-position: left center;
  font-size: 10pt;
  font-weight: bold;
  line-height: 25px;
  padding-left: 20px;
}
```

If vertical scrolling is needed, the scroll bar is rendered in the panel near the right edge of the dialog, which is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogscrollpanel
```

CSS properties of the dialog box scroll panel

width	Scroll panel width.
-------	---------------------

Example – to set up a scroll panel to be 44 pixels wide:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogscrollpanel {
  width: 44px;
}
```

The appearance of the scroll bar area is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar
```

CSS properties of the scroll bar

width	The scroll bar width.
top	The vertical scroll bar offset from the top of the scroll panel.
bottom	The vertical scroll bar offset from the bottom of the scroll panel.
right	The horizontal scroll bar offset from the right edge of the scroll panel.

Example – to set up a scroll bar that is 28 pixels wide, an eight pixel margin from the top, right, and bottom of the scroll panel:

```
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar {
  bottom: 8px;
  right: 8px;
  top: 8px;
  width: 28px;
}
```

Scroll bar track is the area between the top and bottom scroll buttons. The component automatically sets the position and height of the track. The track is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrolltrack
```

CSS properties of the scroll track

width	The track width.
-------	------------------

background-color	The track background color.
------------------	-----------------------------

Example – to set up a scroll bar track that is 28 pixels wide and has a gray background:


```
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrolltrack {
width:28px;
background-color: #B2B2B2;
}
```

Scroll bar thumb moves vertically within a scroll track area. Its vertical position is fully controlled by the component logic, however the thumb height does not dynamically change depending on the amount of content. You can configure the thumb height and other aspects with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrollthumb
```

CSS properties of the scroll bar thumb

width	The thumb width.
height	The thumb height.
padding-top	The vertical padding between the top of the track.
padding-bottom	The vertical padding between the bottom of the track.
background-image	The image that is displayed for a given thumb state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .

 **Note:** Thumb supports the state attribute selector, which can be used to apply different skins to different thumb states: up, down, over, and disabled.

Example – to set up scroll bar thumb that is 28 x 45 pixels, has a ten pixel margin on the top and the bottom, and has different artwork for each state:

```
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrollthumb {
  height: 45px;
  padding-bottom: 10px;
  padding-top: 10px;
  width: 28px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrollthumb[state="up"] {
  background-image:url("images/sdk/scrollbar_thumb_up.png");
}
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrollthumb[state="down"] {
  background-image:url("images/sdk/scrollbar_thumb_down.png");
}
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrollthumb[state="over"] {
  background-image:url("images/sdk/scrollbar_thumb_over.png");
}
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrollthumb[state="disabled"] {
  background-image:url("images/sdk/scrollbar_thumb_disabled.png");
}
```

The appearance of the top and bottom scroll buttons is controlled with the following CSS class selectors:

```
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrollupbutton
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton
```

It is not possible to position scroll buttons using CSS top, left, bottom, and right properties. Instead, the viewer logic automatically positions them.

CSS properties of the top and bottom scroll buttons

width	The button width.
height	The button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: These buttons support the *state* attribute selector, which can be used to apply different skins to different button states: up, down, over, and disabled.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up scroll buttons that are 28 x 32 pixels and have different artwork for each state:

```
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrollupbutton {
  width:28px;
  height:32px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrollupbutton[state='up'] {
  background-image:url(images/sdk/scroll_up_up.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrollupbutton[state='over'] {
  background-image:url(images/sdk/scroll_up_over.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrollupbutton[state='down'] {
  background-image:url(images/sdk/scroll_up_down.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrollupbutton[state='disabled'] {
  background-image:url(images/sdk/scroll_up_disabled.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton {
  width:28px;
  height:32px;
}
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton[state='up'] {
  background-image:url(images/sdk/scroll_down_up.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton[state='over'] {
  background-image:url(images/sdk/scroll_down_over.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton[state='down'] {
  background-image:url(images/sdk/scroll_down_down.png);
}
.s7ecatalogsearchviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton[state='disabled'] {
  background-image:url(images/sdk/scroll_down_disabled.png);
}
```

Embed share

Embed share tool consists of a button added to the Social share panel and the modal dialog box that displays when the tool is activated. The position of the button is fully managed by the Social share tool.

The appearance of the embed share button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embedshare
```

CSS properties of the embed share tool

width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a embed share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```
.s7ecatalogsearchviewer .s7embedshare {
  width:28px;
  height:28px;
}
.s7ecatalogsearchviewer .s7embedshare[state='up'] {
  background-image:url(images/v2/EmbedShare_dark_up.png);
}
.s7ecatalogsearchviewer .s7embedshare[state='over'] {
  background-image:url(images/v2/EmbedShare_dark_over.png);
}
.s7ecatalogsearchviewer .s7embedshare[state='down'] {
  background-image:url(images/v2/EmbedShare_dark_down.png);
}
.s7ecatalogsearchviewer .s7embedshare[state='disabled'] {
  background-image:url(images/v2/EmbedShare_dark_disabled.png);
}
```

The background overlay that covers the web page when the dialog box is active is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7backoverlay
```

CSS properties of the background overlay

opacity	Background overlay opacity.
background-color	Background overlay color.

Example – to set up a background overlay to be gray with 70% opacity:

```
.s7ecatalogsearchviewer .s7embeddialog .s7backoverlay {
  opacity:0.7;
  background-color:#222222;
}
```

By default the modal dialog box is displayed centered in the screen on desktop systems and takes the entire web page area on touch devices. In all cases, the positioning and sizing of the dialog box is managed by the component. The dialog box is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialog
```

CSS properties of the dialog box

border-radius	Dialog box border radius, in case the dialog box does not take the entire browser.
background-color	Dialog box background color.
width	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).
height	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).

Example – to set up the dialog box to use the entire browser window and have a white background on touch devices:

```
.s7ecatalogsearchviewer .s7touchinput .s7embeddialog .s7dialog {
  width:100%;
  height:100%;
  background-color: #ffffff;
}
```

Dialog box header consists of an icon, a title text, and a close button. The header container is controlled with

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogheader
```

CSS properties of the dialog box header

padding	Inner padding for header content.
---------	-----------------------------------

The icon and the title text are wrapped into an additional container controlled with

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogheader .s7dialogline
```

CSS properties of the dialog line

padding	Inner padding for the header icon and title
---------	---

Header icon is controlled with the following CSS class selector

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogheadericon
```

CSS properties of the dialog box header icon

width	Icon width.
-------	-------------

height	Icon height.
background-image	Icon image.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .

Header title is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogheadertext
```

CSS properties of the dialog box header text

font-weight	Font weight.
font-size	Font height.
font-family	Font family.
padding	Internal text padding.

Close button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7closebutton
```

CSS properties of the close button

top	Vertical button position relative to header container.
right	Horizontal button position relative to header container.
width	Button width.
height	Button height.
padding	Inner padding of button.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up dialog header with padding, 24 x 14 pixels icon, bold 16 point title, and 28 x 28 pixels close button, positioned two pixels from the top, and two pixels from the right of dialog container:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogheader {
  padding: 10px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogheader .s7dialogline {
  padding: 10px 10px 2px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogheadericon {
  background-image: url("images/sdk/dlgembed_cap.png");
  height: 14px;
  width: 24px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogheadertext {
  font-size: 16pt;
  font-weight: bold;
  padding-left: 16px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7closebutton {
  top: 2px;
  right: 2px;
  padding: 8px;
  width: 28px;
  height: 28px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7closebutton[state='up'] {
  background-image: url(images/sdk/close_up.png);
}
.s7ecatalogsearchviewer .s7embeddialog .s7closebutton[state='over'] {
  background-image: url(images/sdk/close_over.png);
}
.s7ecatalogsearchviewer .s7embeddialog .s7closebutton[state='down'] {
  background-image: url(images/sdk/close_down.png);
}
.s7ecatalogsearchviewer .s7embeddialog .s7closebutton[state='disabled'] {
  background-image: url(images/sdk/close_disabled.png);
}
```

Dialog footer consists of "cancel" button. The footer container is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogfooter
```

CSS properties of the dialog box footer

border	Border that you can use to visually separate the footer from the rest of the dialog box.
--------	--

The footer has an inner container which keeps the button. It is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogbuttoncontainer
```

CSS properties of the dialog box button container

padding	Inner padding between the footer and the button.
---------	--

Select All button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogactionbutton
```

The button is only available on desktop systems.

CSS properties of the Select All button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: The *Select All* button supports the *state* attribute selector, which can be used to apply different skins to different button states.

Cancel button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogcancelbutton
```

CSS properties of the dialog box cancel button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

In addition, both buttons share the same common CSS class which can contain CSS settings that are the same for other dialog box buttons:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogfooter .s7button
```

CSS properties of the button

font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.
line-height	Text height inside the button. Affects vertical alignment.
box-shadow	Drop shadow.
margin-right	Right button margin.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box footer with a 64 x 34 Cancel button, an 82 x 34 Select All button, and having a text color and background color that is different for each button state:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogfooter {
  border-top: 1px solid #909090;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogbuttoncontainer {
  padding-bottom: 6px;
  padding-top: 10px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogfooter .s7button {
  box-shadow: 1px 1px 1px #999999;
  color: #FFFFFF;
  font-size: 9pt;
  font-weight: bold;
  line-height: 34px;
  margin-right: 10px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogcancelbutton {
  width: 64px;
  height: 34px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogcancelbutton[state='up'] {
  background-color: #666666;
  color: #dddddd;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogcancelbutton[state='down'] {
  background-color: #555555;
  color: #ffffff;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogcancelbutton[state='over'] {
  background-color: #555555;
  color: #ffffff;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogcancelbutton[state='disabled'] {
  background-color: #b2b2b2;
  color: #dddddd;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogactionbutton {
  width: 82px;
  height: 34px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogactionbutton[state='up'] {
  background-color: #333333;
  color: #dddddd;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogactionbutton[state='down'] {
  background-color: #222222;
  color: #cccccc;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogactionbutton[state='over'] {
  background-color: #222222;
  color: #cccccc;
}
.s7ecatalogsearchviewer .s7embeddialog .s7dialogactionbutton[state='disabled'] {
  background-color: #b2b2b2;
  color: #dddddd;
}
```

The main dialog area (between the header and the footer) contains scrollable dialog content and scroll panel on the right. In all cases, the component manages the width of this area, it is not possible to set it in CSS. Main dialog area is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogviewarea
```

CSS properties of the dialog box viewing area

height	The height of the main dialog box area. It should be specified only when the dialog box works in desktop mode. It is not applicable when the dialog box is sized to occupy the entire browser window.
background-color	The background color of the main dialog box area.
margin	Outer margin.

Example – to set up a main dialog box area to be 300 pixels height, have a ten pixel margin, and use a white background:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogviewarea {
  background-color:#ffffff;
  margin:10px;
  height:300px;
}
```

All form content (like labels and input fields) resides inside a container controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogbody
```

If the height of this container appears to be bigger than the main dialog box area, a vertical scroll is enabled automatically by the component.

CSS properties of the dialog box body

padding	Inner padding.
---------	----------------

Example – to set up form content to have ten pixel padding:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogbody {
  padding: 10px;
}
```

All static labels in the dialog box form are controlled with

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialoglabel
```

This class is not suitable for controlling the label size or position because you can apply it to texts in various places of the form user interface.

CSS properties of the dialog box label.

font-weight	Label font weight.
font-size	Label font size.
font-family	Label font family.
color	Label text color.

Dialog box labels can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up all labels to be gray, bold with a nine pixel font:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialoglabel {
  color: #666666;
  font-size: 9pt;
  font-weight: bold;
}
```

The size of the text copy displayed on top of the embed code is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialoginputwide
```

CSS properties of the dialog box input wide field

width	Input field width.
padding	Inner padding.

Example – to set text copy to be 430 pixels wide and have a ten pixel padding in the bottom:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialoginputwide {
  padding-bottom: 10px;
  width: 430px;
}
```

The embed code is wrapped into container and controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

width	The width of the embed code container.
border	Border around the embed code container.
padding	Inner padding.

Example – to set a one pixel grey border around embed code text, make it 430 pixels wide, and have a ten pixel padding:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialoginputcontainer {
  border: 1px solid #CCCCCC;
  padding: 10px;
  width: 430px;
}
```

The actual embed code text is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

word-wrap	Word wrapping style.
-----------	----------------------

Example – to set up embed code to use break-word word wrapping:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogmessage {
  word-wrap: break-word;
}
```

Embed size label and drop-down are located in the bottom of the dialog box and put into a container controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogembedsizepanel
```

CSS properties of the dialog box embed size panel

padding	Inner padding.
---------	----------------

Example – to set up an embed size panel to have ten pixels of padding:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogembedsizepanel {
  padding: 10px;
}
```

The size and alignment of the embed size label is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogembedsizelabel
```

CSS properties of the dialog box embed size panel

vertical-align	Vertical label alignment.
width	Label width.

Example – to set the embed size label to be top-aligned and 80 pixels wide:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogembedsizelabel {
  vertical-align: top;
  width: 80px;
}
```

The width of the embed size combo box is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7combobox
```

CSS properties of the combo box

width	Combo box width.
-------	------------------



Note: The combo box supports the *expanded* attribute selector with possible values of *true* and *false*. *true* is used when combo box displays one of pre-defined embed sizes, thus should take all available width. *false* is used when custom size option is selected in the combo box, so it should shrink to allow space for custom width and height input fields.

Example – to set the embed size combo box to be 300 pixels wide when showing a pre-defined item and 110 pixels wide when showing a custom size:

```
.s7ecatalogsearchviewer .s7embeddialog .s7combobox[expanded="true"] {
  width: 300px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7combobox[expanded="false"] {
  width: 110px;
}
```

The height of the combo box text is defined by a special inner element and is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7combobox .s7comboboxtext
```

CSS properties of the combo box text

height	Combo box text height.
--------	------------------------

Example – to set embed size combo box text height to 40 pixels:

```
.s7ecatalogsearchviewer .s7embeddialog .s7combobox .s7comboboxtext {
    height: 40px;
}
```

The combo box has a "drop down" button on the right and it is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7combobox .s7comboboxbutton
```

CSS properties of the combo box button

top	Vertical button position inside the combo box.
right	Horizontal button position inside the combo box.
width	Button width.
height	Button height.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

Example – to set a drop-down button to 28 x 28 pixels and have a separate image for each state:

```
.s7ecatalogsearchviewer .s7embeddialog .s7combobox .s7comboboxbutton {
    width:28px;
    height:28px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7combobox .s7comboboxbutton[state='up'] {
    background-image:url(images/sdk/cboxbtndn_up.png);
}
.s7ecatalogsearchviewer .s7embeddialog .s7combobox .s7comboboxbutton[state='over'] {
    background-image:url(images/sdk/cboxbtndn_over.png);
}
.s7ecatalogsearchviewer .s7embeddialog .s7combobox .s7comboboxbutton[state='down'] {
    background-image:url(images/sdk/cboxbtndn_over.png);
}
.s7ecatalogsearchviewer .s7embeddialog .s7combobox .s7comboboxbutton[state='disabled'] {
    background-image:url(images/sdk/cboxbtndn_up.png);
}
```

The panel with the list of embed sizes displayed when combo box is opened is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7comboboxdropdown
```

The size and position of the panel is controlled by the component. It is not possible to change it through CSS.

CSS properties of the combo box drop-down

border	Panel border.
--------	---------------

Example – to set the combo box panel to have a one pixel grey border:

```
.s7ecatalogsearchviewer .s7embeddialog .s7comboboxdropdown {
  border: 1px solid #CCCCCC;
}
```

A single item in a drop-down panel that is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dropdownitemanchor
```

CSS properties of the drop-down item anchor

background-color	Item background.
------------------	------------------

Example – to set the combo box panel item to have a white background:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dropdownitemanchor {
  background-color: #FFFFFF;
}
```

A check mark displayed to the left of the selected item inside the combo box panel that is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7checkmark
```

CSS properties of the check mark box

width	Icon width.
height	Icon height.
background-image	Item image.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .

Example – to set the check mark icon to 25 x 25 pixels:

```
.s7ecatalogsearchviewer .s7embeddialog .s7checkmark {
  background-image: url("images/sdk/cboxchecked.png");
  height: 25px;
  width: 25px;
}
```

When "Custom Size" option is selected in the embed size combo box the dialog box displays two extra input fields to the right to allow the user to enter a custom embed size. Those fields are wrapped in a container that is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogcustomsizepanel
```

CSS properties of the dialog box custom size panel

left	Distance from the embed size combo box.
------	---

Example – to set custom size input fields panel to be 20 pixels to the right of the combo box:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogcustomsizepanel {
  left: 20px;
}
```

Each custom size input field is wrapped in a container that renders a border and sets the margin between the fields. It is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogcustomsize
```

CSS properties of the dialog box custom size

border	Border around the input field.
width	Input field width.
margin	Input field margin.
padding	Input field padding.

Example – to set the custom size input fields to have a one pixel grey border, margin, padding and be 70 pixels wide:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogcustomsize {
  border: 1px solid #CCCCCC;
  margin-right: 20px;
  padding-left: 2px;
  padding-right: 2px;
  width: 70px;
}
```

If vertical scrolling is needed, the scroll bar is rendered in the panel near the right edge of the dialog box, which is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogscrollpanel
```

CSS properties of the dialog box scroll panel

width	Scroll panel width.
-------	---------------------

Example – to set up a scroll panel to be 44 pixels wide

```
.s7ecatalogsearchviewer .s7embeddialog .s7dialogscrollpanel {
  width: 44px;
}
```

The appearance of the scroll bar area is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar
```

CSS properties of the scroll bar

width	Scroll bar width.
top	The vertical scroll bar offset from the top of the scroll panel.
bottom	The vertical scroll bar offset from the bottom of the scroll panel.

right	The horizontal scroll bar offset from the right edge of the scroll panel.
-------	---

Example – to set up a scroll bar that is 28 pixels wide and has an eight pixel margin from the top, right, and bottom of the scroll panel:

```
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar {
  bottom: 8px;
  right: 8px;
  top: 8px;
  width: 28px;
}
```

Scroll bar track is the area between the top and bottom scroll buttons. The component automatically sets the position and height of the track. The track is controlled with the following CSS class selector

```
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrolltrack
```

CSS properties of the scroll bar track

width	Track width.
background-color	Track background color.

Example – to set up a scroll bar track that is 28 pixels wide and has a grey background:

```
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrolltrack {
width:28px;
background-color: #B2B2B2;
}
```

The scroll bar thumb moves vertically within a scroll track area. Its vertical position is fully controlled by the component logic. However, thumb height does not dynamically change depending on the amount of content. The thumb height and other aspects can be configured with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrollthumb
```

CSS properties of the scroll bar thumb

width	Thumb width.
height	Thumb height.
padding-top	The vertical padding between the top of the track.
padding-bottom	The vertical padding between the bottom of the track.
background-image	The image displayed for a given thumb state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: Thumb supports the `state` attribute selector, which can be used to apply different skins to different thumb states: up, down, over, and disabled.

Example – to set up a scroll bar thumb that is 28 x 45 pixels, has a ten pixel margin on the top and bottom, and has different artwork for each state:

```
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrollthumb {
  height: 45px;
  padding-bottom: 10px;
  padding-top: 10px;
  width: 28px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrollthumb[state="up"] {
  background-image:url("images/sdk/scrollbar_thumb_up.png");
}
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrollthumb[state="down"] {
  background-image:url("images/sdk/scrollbar_thumb_down.png");
}
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrollthumb[state="over"] {
  background-image:url("images/sdk/scrollbar_thumb_over.png");
}
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrollthumb[state="disabled"] {
  background-image:url("images/sdk/scrollbar_thumb_disabled.png");
}
```

The appearance of the top and bottom scroll buttons is controlled with the following CSS class selectors:

```
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrollupbutton
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton
```

It is not possible to position scroll buttons using CSS `top`, `left`, `bottom`, and `right` properties. Instead, the viewer logic automatically positions them.

CSS properties of the top and bottom scroll buttons

width	Button width.
height	Button height.
background-image	The image displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: These buttons support the `state` attribute selector, which can be used to apply different skins to different button states: up, down, over, and disabled.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up scroll buttons that are 28 x 32 pixels and have different artwork for each state:

```
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrollupbutton {
  width:28px;
  height:32px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrollupbutton[state='up'] {
  background-image:url(images/sdk/scroll_up_up.png);
}
```

```

}
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrollupbutton[state='over'] {
  background-image:url(images/sdk/scroll_up_over.png);
}
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrollupbutton[state='down'] {
  background-image:url(images/sdk/scroll_up_down.png);
}
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrollupbutton[state='disabled'] {
  background-image:url(images/sdk/scroll_up_disabled.png);
}
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton {
  width:28px;
  height:32px;
}
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton[state='up'] {
  background-image:url(images/sdk/scroll_down_up.png);
}
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton[state='over'] {
  background-image:url(images/sdk/scroll_down_over.png);
}
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton[state='down'] {
  background-image:url(images/sdk/scroll_down_down.png);
}
.s7ecatalogsearchviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton[state='disabled'] {
  background-image:url(images/sdk/scroll_down_disabled.png);
}
}

```

Facebook share

Facebook share tool consists of a button added to the Social share panel. When the button is clicked the user is redirected to a sharing dialog box that is provided by a social service. The position of the button is fully managed by the Social share tool.

The appearance of the Facebook share button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7facebookshare
```

CSS properties of the Facebook share tool

width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Facebook share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```

.s7ecatalogsearchviewer .s7facebookshare {
  width:28px;
  height:28px;
}

```

```
}
.s7ecatalogsearchviewer .s7facebookshare[state='up'] {
background-image:url(images/v2/FacebookShare_dark_up.png);
}
.s7ecatalogsearchviewer .s7facebookshare[state='over'] {
background-image:url(images/v2/FacebookShare_dark_over.png);
}
.s7ecatalogsearchviewer .s7facebookshare[state='down'] {
background-image:url(images/v2/FacebookShare_dark_down.png);
}
.s7ecatalogsearchviewer .s7facebookshare[state='disabled'] {
background-image:url(images/v2/FacebookShare_dark_disabled.png);
}
```

Favorites effect

The viewer displays Favorites icons over the main view in places where it was originally added by the user.

The appearance of the Favorite icon is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7favoriteseffect .s7icon
```

CSS properties of the Favorite icon

background-image	The image that is displayed for the icon.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
width	Width of the icon.
height	Height of the icon.

Example – set up a 36 x 36 pixels Favorites icon.

```
.s7ecatalogsearchviewer .s7favoriteseffect .s7icon {
height: 36px;
width: 36px;
background-image: url(images/v2/FavoriteEffect_dark_up.png);
}
```

On desktop systems, the component supports the `cursor` attribute selector which you can apply to the `.s7favoriteseffect` class and controls the type of the cursor based on the selected user action. The following `cursor` values are supported:

mode_add	Displayed user is adding a new Favorite icon.
mode_remove	Displayed user is removing an existing Favorite icon.
mode_view	Displayed in normal operation mode when Favorites editing is not active.

Example – have different mouse cursors for each type of component state.

```
.s7ecatalogsearchviewer .s7favoriteseffect[cursor="mode_add"] {
cursor: crosshair;
```

```

}
.s7ecatalogsearchviewer .s7favoriteseffect[cursortype="mode_remove"] {
cursor: not-allowed;
}
.s7ecatalogsearchviewer .s7favoriteseffect[cursortype="mode_view"] {
cursor: auto;
}

```

Favorites menu

The Favorites menu drop-down list appears in the control bar. It consists of a button and a panel that expands when a user clicks or taps on a button. The panel contains individual Favorites tools.

The position and size of the Favorites menu in the viewer user interface is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7favoritesmenu
```

CSS properties of the Favorites menu button

margin-top	The offset from the top of the control bar.
margin-left	The distance to the next button on the left, or the left side of the control bar if this is the first button in a row.
width	Width of the button.
height	Height of the button.

Example – set up a Favorites menu that is positioned four pixels from the top of the control bar and ten pixels from the closest button to the left and sized 28 x 28 pixels.

```

.s7ecatalogsearchviewer .s7favoritesmenu {
margin-top: 4px;
margin-left: 10px;
width: 28px;
height: 28px;
}

```

The appearance of the Favorites menu button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7favoritesmenu .s7favoritesbutton
```

CSS properties of the Favorites button

background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – set up a Favorites menu button that displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7favoritesmenu .s7favoritesbutton[state='up'] {
background-image:url(images/v2/FavoritesMenu_dark_up.png);
}
.s7ecatalogsearchviewer .s7favoritesmenu .s7favoritesbutton[state='over'] {
background-image:url(images/v2/FavoritesMenu_dark_over.png);
}
.s7ecatalogsearchviewer .s7favoritesmenu .s7favoritesbutton[state='down'] {
background-image:url(images/v2/FavoritesMenu_dark_down.png);
}
.s7ecatalogsearchviewer .s7favoritesmenu .s7favoritesbutton[state='disabled'] {
background-image:url(images/v2/FavoritesMenu_dark_disabled.png);
}
```

The appearance of the panel that contains individual Favorites icons is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7favoritesmenu .s7favoritesmenupanel
```

CSS properties of the Favorites menu panel

background-color	The background color of the panel.
------------------	------------------------------------

Example – set up a panel to have a transparent color.

```
.s7ecatalogsearchviewer .s7favoritesmenu .s7favoritesmenupanel {
background-color: transparent;
}
```

Favorites view

Favorites view consist of a column of thumbnail images.

The appearance of favorites view container is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7favoritesview
```

The position and the height of the Favorites view is managed by the view; in CSS it is only possible to define the width.

CSS properties of the Favorites view

background-color	Background color of the Favorites view.
width	Width of the view.

Example – to set up a Favorites view that is 100 pixels wide with a semi-transparent grey background.

```
.s7ecatalogsearchviewer .s7favoritesview {
width: 100px;
background-color: rgba(221, 221, 221, 0.5);
}
```

The spacing between Favorites thumbnails is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7favoritesview .s7thumbcell
```

CSS properties of the Favorites thumbnails

margin	The size of the vertical margin around each thumbnail. The actual thumbnail spacing is equal to the sum of the top and bottom margin that is set for .s7thumbcell.
--------	--

Example – to set up 10 pixel spacing.

```
.s7ecatalogsearchviewer .s7favoritesview .s7thumbcell {
  margin: 5px;
}
```

The appearance of individual thumbnail is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7favoritesview .s7thumb
```

CSS properties of the Favorites thumbnails

width	Width of the thumbnail.
height	Height of the thumbnail.
border	Border of the thumbnail.



Note: Thumbnail supports the *state* attribute selector, which can be used to apply different skins to different thumbnail states. In particular, *state="selected"* corresponds to the thumbnail recently selected by the user. *state="default"* corresponds to the rest of the thumbnails. And *state="over"* is used on mouse hover.

Example – to set up thumbnails that are 75 x 75 pixels, have a light grey default border, and a dark grey selected border.

```
.s7ecatalogsearchviewer .s7favoritesview .s7thumb {
  width: 75px;
  height: 75px;
}
.s7ecatalogsearchviewer .s7favoritesview .s7thumb[state="default"] {
  border: 1px solid #dddddd;
}
.s7ecatalogsearchviewer .s7favoritesview .s7thumb[state="selected"] {
  border: 1px solid #666666;
}
```

The appearance of the thumbnail label is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7favoritesview .s7label
```

CSS properties of the Favorites label

font-family	Font name.
font-size	Font size.

Example – to set up labels with a 14 pixel Helvetica font.

```
.s7ecatalogsearchviewer .s7favoritesview .s7label {
  font-family: Helvetica,sans-serif;
  font-size: 14px;
}
```

First page button

Clicking or tapping on this button brings the user to the first page in the catalog. This button appears in the main control bar on desktop systems and tablets; on mobile phones it is added to a secondary control bar. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7firstpagebutton .s7panleftbutton
```

CSS property	Description
top	Position from the top border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
right	Position from the right border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
left	Position from the left border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
bottom	Position from the bottom border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a first page button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 220 pixels from the left edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7firstpagebutton .s7panleftbutton {
bottom:4px;
left:220px;
width:32px;
height:32px;
}
.s7ecatalogsearchviewer .s7firstpagebutton .s7panleftbutton [state='up'] {
background-image:url(images/v2/FirstPageButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7firstpagebutton .s7panleftbutton [state='over'] {
background-image:url(images/v2/FirstPageButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7firstpagebutton .s7panleftbutton [state='down'] {
background-image:url(images/v2/FirstPageButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7firstpagebutton .s7panleftbutton [state='disabled'] {
```

```
background-image:url(images/v2/FirstPageButton_dark_disabled.png);
}
```

Full screen button

Causes the viewer to enter or exit full screen mode when clicked by the user. This button appears in the main control bar. This button is not displayed if the viewer works in pop-up mode and the system does not support native full screen. You can size, skin, and positioned the button by CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7fullscreenbutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.
left	Position from the left border of the main control bar, including padding.
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the "full screen" state and *selected='false'* corresponds to the "normal" state.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a full screen button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 5 pixels from the right edge of the main control bar, and displays a different image for each of the four different button states when selected or not selected.

```
.s7ecatalogsearchviewer .s7fullscreenbutton {
bottom:4px;
right:5px;
width:28px;
height:28px;
}
.s7ecatalogsearchviewer .s7fullscreenbutton [selected='false'][state='up'] {
background-image:url(images/enterFullBtn_up.png);
}
```

```

}
.s7ecatalogsearchviewer .s7fullscreenbutton [selected='false'][state='over'] {
background-image:url(images/enterFullBtn_over.png);
}
.s7ecatalogsearchviewer .s7fullscreenbutton [selected='false'][state='down'] {
background-image:url(images/enterFullBtn_down.png);
}
.s7ecatalogsearchviewer .s7fullscreenbutton [selected='false'][state='disabled'] {
background-image:url(images/enterFullBtn_disabled.png);
}
.s7ecatalogsearchviewer .s7fullscreenbutton [selected='true'][state='up'] {
background-image:url(images/exitFullBtn_up.png);
}
.s7ecatalogsearchviewer .s7fullscreenbutton [selected='true'][state='over'] {
background-image:url(images/exitFullBtn_over.png);
}
.s7ecatalogsearchviewer .s7fullscreenbutton [selected='true'][state='down'] {
background-image:url(images/exitFullBtn_down.png);
}
.s7ecatalogsearchviewer .s7fullscreenbutton [selected='true'][state='disabled'] {
background-image:url(images/exitFullBtn_disabled.png); }
}

```

Icon effect

The zoom indicator is overlaid on the main view area. It is displayed when the image is in a reset state and it also depends on `iconeffect` parameter.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7pageview .s7iconeffect
```

CSS property	Description
background-image	Zoom indicator artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
width	Zoom indicator width in pixels.
height	Zoom indicator height in pixels.



Note: Icon effect supports the `media-type` attribute selector, which you can use to apply different icon effects on different devices. In particular, `media-type='standard'` corresponds to desktop systems where mouse input is normally used and `media-type='multitouch'` corresponds to devices with touch input.

Example – to set up a 100 x 100 pixel zoom indicator with different art for desktop systems and touch devices.

```

.s7ecatalogsearchviewer .s7pageview .s7iconeffect {
width: 100px;
height: 100px;
}
.s7ecatalogsearchviewer .s7pageview .s7iconeffect[media-type='standard'] {
background-image:url(images/v2/IconEffect_zoom.png);
}

```

```
.s7ecatalogsearchviewer .s7pageview .s7iconeffect[media-type='multitouch'] {
  background-image:url(images/v2/IconEffect_pinch.png);
}
```

Info panel popup

Info Panel Popup displays in the middle of the viewer area when a user activates an image map that has a rollover_key property defined in Scene7 Publishing System, and if info panel feature is properly configured for the viewer.

Info panel background covers entire viewer area and is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7infopanelpopup .s7backoverlay
```

CSS property	Description
background-image	Info panel background fill.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .

Example – set up the info panel popup to use a semi-transparent black background.

```
.s7ecatalogsearchviewer .s7infopanelpopup .s7backoverlay {
  background-color : rgba(0,0,0,0.75);
}
```

The info panel dialog is displayed by default in the middle of the viewer area. However it is possible to control its size, alignment, background, and border with the CSS class selector.

```
.s7ecatalogsearchviewer .s7infopanelpopup .s7overlay
```

CSS property	Description
left	Horizontal position of the info panel dialog within viewer area panel background fill.
top	Vertical position of the info panel dialog within viewer area.
width	Dialog width.
height	Dialog height.
margin-left	Left margin of the info panel dialog, may be used for centering purposes.
margin-top	Top margin of the info panel dialog, may be used for centering purposes.
padding	Internal dialog padding.
background-color	Dialog background color.

CSS property	Description
border-radius	Dialog border radius.
box-shadow	Dialog shadow.

Example – set up 300 x 200 pixels info panel dialog that is centered in the viewer area; has 40 pixels padding at the top and 10 pixels padding on all other sides, a light gray background, and a 10 pixel border radius and drop shadow.


```
.s7ecatalogsearchviewer .s7infopanelpopup .s7overlay {
  left: 50%;
  top: 50%;
  margin-left: -150px;
  margin-top: -100px;
  width: 300px;
  height: 200px;
  padding-top: 40px;
  padding-right: 10px;
  padding-bottom: 10px;
  padding-left: 10px;
  background-color:rgb(221,221,221);
  border-radius: 10px 10px 10px 10px;
  box-shadow: 0 0 5px rgba(0,0,0,0.25);
}
```

The Info Panel dialog has a close button, and clicking or tapping the button closes the dialog.

The appearance of this button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7infopanelpopup .s7closebutton
```

CSS property	Description
top	Position from the top border of the dialog.
right	Position from the right border of the dialog.
left	Position from the left border of the dialog.
bottom	Position from the bottom border of the dialog.
width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .

 **Note:** This button supports the `state` attribute selector, which you can use to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog close button that is 28 x 28 pixels, positioned 5 pixels from the top and right edge of the info panel dialog, and displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7infopanelpopup .s7closebutton {
  width: 28px;
  height: 28px;
  top: 5px;
  right: 5px;
}
.s7ecatalogsearchviewer .s7infopanelpopup .s7closebutton[state="up"] {
  background-image:url(images/v2/InfoPanelPopup_CloseButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7infopanelpopup .s7closebutton[state="over"] {
  background-image:url(images/v2/InfoPanelPopup_CloseButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7infopanelpopup .s7closebutton[state="down"] {
  background-image:url(images/v2/InfoPanelPopup_CloseButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7infopanelpopup .s7closebutton[state="disabled"] {
  background-image:url(images/v2/InfoPanelPopup_CloseButton_dark_up.png);
}
```


Image map effect

Depending on the value of the `mode` parameter, the viewer displays image map icons over the main view in places where maps are originally authored in Scene7 Publishing System or renders exact regions that match the shape of original image maps.


CSS properties of the main viewer area

The appearance of the image map icon is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7imagemapeffect .s7icon
```

 **Note:** The `s7mapoverlay` CSS class used to style image map icons in the past is now deprecated; use `s7icon` instead.

CSS property	Description
background-image	Image map icon artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
width	Image map icon width in pixels.
height	Image map icon height in pixels.

 **Note:** Image map icon supports the `state` attribute selector, which you can use to apply different skins to the icon states of default and active.

Example – set up a 28 x 28 pixels image map icon that displays a different image for each of the two different icon states.

```
.s7ecatalogsearchviewer .s7imagemapeffect .s7icon {
  height: 28px;
  width: 28px;
  background-image: url(images/v2/ImageMapEffect_dark_up.png);
}
.s7ecatalogsearchviewer .s7imagemapeffect .s7icon[state="default"] {
  opacity: 0.5;
}
.s7ecatalogsearchviewer .s7imagemapeffect .s7icon[state="active"] {
  opacity: 1;
}
```

See also [Image map support](#).

The appearance of the image map region is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7imagemapeffect .s7region
```

CSS property	Description
background	Image map region fill color. Specified in #RRGGBB, RGB(R,G,B), or RGBA(R,G,B,A) format.
background-color	Image map region fill color. Specified in #RRGGBB, RGB(R,G,B) or RGBA(R,G,B,A) format.
border	Image map region border style. Specified as <i>width solid color</i> , where <i>width</i> is expressed in pixels and <i>color</i> is set as #RRGGBB, RGB(R,G,B) or RGBA(R,G,B,A).

Example – set up a transparent image map region with 1 pixel black border :

```
.s7ecatalogsearchviewer .s7imagemapeffect .s7region {
  border: 1px solid #000000;
  background: RGBA(0,0,0,0);
}
```

Large next page button

Clicking or tapping on this button brings the user to the next page in the catalog. This button appears in the main control bar. This button is not displayed on mobile phones to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7ecatrightbutton .s7panrightbutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.

CSS property	Description
right	Position from the right border of the main control bar, including padding.
left	Position from the left border of the main control bar, including padding.
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a large next page button that is 56 x 56 pixels, vertically centered and anchored to the right viewer border, and displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7ecatrightbutton .s7panrightbutton {
bottom:50%;
margin-bottom:-28px;
right:0px;
width:56px;
height:56px;
}
.s7ecatalogsearchviewer .s7ecatrightbutton .s7panrightbutton [state='up'] {
background-image:url(images/v2/RightButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7ecatrightbutton .s7panrightbutton [state='over'] {
background-image:url(images/v2/RightButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7ecatrightbutton .s7panrightbutton [state='down'] {
background-image:url(images/v2/RightButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7ecatrightbutton .s7panrightbutton [state='disabled'] {
background-image:url(images/v2/RightButton_dark_disabled.png);
}
```

Large previous page button

Clicking or tapping on this button brings the user to the previous page in the catalog. This button appears in the main control bar. This button is not displayed on mobile phones to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

.s7ecatalogsearchviewer .s7ecatleftbutton .s7panleftbutton

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.
left	Position from the left border of the main control bar, including padding.
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a large previous page button that is 56 x 56 pixels, vertically centered and anchored to the left viewer border, and displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7ecatleftbutton .s7panleftbutton {
bottom:50%;
margin-bottom:-28px;
left:0px;
width:56px;
height:56px;
}
.s7ecatalogsearchviewer .s7ecatleftbutton .s7panleftbutton [state='up'] {
background-image:url(images/v2/LeftButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7ecatleftbutton .s7panleftbutton [state='over'] {
background-image:url(images/v2/LeftButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7ecatleftbutton .s7panleftbutton [state='down'] {
background-image:url(images/v2/LeftButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7ecatleftbutton .s7panleftbutton [state='disabled'] {
background-image:url(images/v2/LeftButton_dark_disabled.png);
}
```

Last page button

Clicking or tapping on this button brings the user to the last page in the catalog. This button appears in the main control bar on desktop systems and tablets; on mobile phones it is added to a secondary control bar. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7lastpagebutton .s7panleftbutton
```

CSS property	Description
top	Position from the top border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
right	Position from the right border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
left	Position from the left border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
bottom	Position from the bottom border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a last page button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 220 pixels from the left edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7lastpagebutton .s7panrightbutton {
bottom:4px;
right:220px;
width:32px;
height:32px;
}
.s7ecatalogsearchviewer .s7lastpagebutton .s7panrightbutton [state='up'] {
background-image:url(images/v2/LastPageButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7lastpagebutton .s7panrightbutton [state='over'] {
background-image:url(images/v2/LastPageButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7lastpagebutton .s7panrightbutton [state='down'] {
background-image:url(images/v2/LastPageButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7lastpagebutton .s7panrightbutton [state='disabled'] {
```

```
background-image:url(images/v2/LastPageButton_dark_disabled.png);
}
```

Link share

Link share tool consists of a button added to the Social share panel and the modal dialog box that displays when the tool is activated. The position of the button is fully managed by the Social share tool.

The appearance of the link share button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkshare
```

CSS properties of the link share tool

width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a link share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```
.s7ecatalogsearchviewer .s7linkshare {
  width:28px;
  height:28px;
}
.s7ecatalogsearchviewer .s7linkshare[state='up'] {
background-image:url(images/v2/LinkShare_dark_up.png);
}
.s7ecatalogsearchviewer .s7linkshare[state='over'] {
background-image:url(images/v2/LinkShare_dark_over.png);
}
.s7ecatalogsearchviewer .s7linkshare[state='down'] {
background-image:url(images/v2/LinkShare_dark_down.png);
}
.s7ecatalogsearchviewer .s7linkshare[state='disabled'] {
background-image:url(images/v2/LinkShare_dark_disabled.png);
}
```

The background overlay that covers the web page when the dialog box is active is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkdialog .s7backoverlay
```

CSS properties of the background overlay

opacity	Background overlay opacity.
---------	-----------------------------

background-color	Background overlay color.
------------------	---------------------------

Example – to set up a background overlay to be gray with 70% opacity:

```
.s7ecatalogsearchviewer .s7linkdialog .s7backoverlay {
  opacity:0.7;
  background-color:#222222;
}
```

By default the modal dialog box is displayed centered in the screen on desktop systems and takes the entire web page area on touch devices. In all cases, the positioning and sizing of the dialog box is managed by the component. The dialog box is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialog
```

CSS properties of the dialog box

border-radius	Dialog box border radius, in case the dialog box does not take the entire browser.
background-color	Dialog box background color.
width	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).
height	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).

Example – to set up the dialog box to use the entire browser window and have a white background on touch devices:

```
.s7ecatalogsearchviewer .s7touchinput .s7linkdialog .s7dialog {
  width:100%;
  height:100%;
  background-color: #ffffff;
}
```

Dialog box header consists of an icon, a title text, and a close button. The header container is controlled with

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogheader
```

CSS properties of the dialog box header

padding	Inner padding for header content.
---------	-----------------------------------

The icon and the title text are wrapped into an additional container controlled with

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogheader .s7dialogline
```

CSS properties of the dialog line

padding	Inner padding for the header icon and title.
---------	--

Header icon is controlled with the following CSS class selector

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogheadericon
```

CSS properties of the dialog box header icon

width	Icon width.
height	Icon height.
background-image	Icon image.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .

Header title is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogheadertext
```

CSS properties of the dialog box header text

font-weight	Font weight.
font-size	Font height.
font-family	Font family.
padding	Internal text padding.

Close button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkdialog .s7closebutton
```

CSS properties of the close button

top	Vertical button position relative to header container.
right	Horizontal button position relative to header container.
width	Button width.
height	Button height.
padding	Inner padding of button.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The Close button tool tip and the dialog box title can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box header with padding, 22 x 12 pixels icon, bold 16 point title, and a 28 x 28 pixel Close button that is positioned two pixels from the top and two pixels from the right of the dialog box container:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogheader {
padding: 10px;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogheader .s7dialogline {
padding: 10px 10px 2px;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogheadericon {
background-image: url("images/sdk/dlglink_cap.png");
height: 12px;
width: 22px;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogheadertext {
font-size: 16pt;
font-weight: bold;
padding-left: 16px;
}
.s7ecatalogsearchviewer .s7linkdialog .s7closebutton {
top: 2px;
right: 2px;
padding: 8px;
width: 28px;
height: 28px;
}
.s7ecatalogsearchviewer .s7linkdialog .s7closebutton[state='up'] {
background-image: url(images/sdk/close_up.png);
}
.s7ecatalogsearchviewer .s7linkdialog .s7closebutton[state='over'] {
background-image: url(images/sdk/close_over.png);
}
.s7ecatalogsearchviewer .s7linkdialog .s7closebutton[state='down'] {
background-image: url(images/sdk/close_down.png);
}
.s7ecatalogsearchviewer .s7linkdialog .s7closebutton[state='disabled'] {
background-image: url(images/sdk/close_disabled.png);
}
```

Dialog box footer consists of a Cancel button. The footer container is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogfooter
```

CSS properties of the dialog box footer

border	Border that you can use to visually separate the footer from the rest of the dialog box.
--------	--

The footer has an inner container which keeps the button. It is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogbuttoncontainer
```

CSS properties of the dialog box button container

padding	Inner padding between the footer and the button.
---------	--

Select All button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogactionbutton
```

The button is only available on desktop systems.

CSS properties of the Select All button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: The Select All button supports the *state* attribute selector, which can be used to apply different skins to different button states.

Cancel button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogcancelbutton
```

CSS properties of the dialog box cancel button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

In addition, both buttons share the same common CSS class which can contain CSS settings that are the same for other dialog box buttons:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogfooter .s7button
```

CSS properties of the button

font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.
line-height	Text height inside the button. Affects vertical alignment.
box-shadow	Drop shadow.
margin-right	Right button margin.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box footer with a 64 x 34 Cancel button, having text color and background color different for each button state:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogfooter {
  border-top: 1px solid #909090;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogbuttoncontainer {
  padding-bottom: 6px;
  padding-top: 10px;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogfooter .s7button {
  box-shadow: 1px 1px 1px #999999;
  color: #FFFFFF;
  font-size: 9pt;
  font-weight: bold;
  line-height: 34px;
  margin-right: 10px;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogcancelbutton {
  width: 64px;
  height: 34px;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogcancelbutton[state='up'] {
  background-color: #666666;
  color: #dddddd;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogcancelbutton[state='down'] {
  background-color: #555555;
  color: #ffffff;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogcancelbutton[state='over'] {
  background-color: #555555;
  color: #ffffff;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogcancelbutton[state='disabled'] {
  background-color: #b2b2b2;
  color: #dddddd;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogactionbutton {
  width: 82px;
  height: 34px;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogactionbutton[state='up'] {
  background-color: #333333;
  color: #dddddd;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogactionbutton[state='down'] {
  background-color: #222222;
  color: #cccccc;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogactionbutton[state='over'] {
  background-color: #222222;
  color: #cccccc;
}
.s7ecatalogsearchviewer .s7linkdialog .s7dialogactionbutton[state='disabled'] {
  background-color: #b2b2b2;
  color: #dddddd;
}
```

The main dialog area (between the header and the footer) contains dialog content. In all cases, the component manages the width of this area—it is not possible to set it in CSS. Main dialog area is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogviewarea
```

CSS properties of the dialog box viewing area

height	The height of the main dialog box area. It should be specified only when the dialog box works in desktop mode. It is not applicable when the dialog box is sized to occupy the entire browser window.
background-color	The background color of the main dialog box area.
margin	Outer margin.

Example – to set up a main dialog box area to be 300 pixels height, have a ten pixel margin, and use a white background:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogviewarea {
  background-color:#ffffff;
  margin:10px;
  height:300px;
}
```

All form content (like labels and input fields) resides inside a container controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogbody
```

CSS properties of the dialog box body

padding	Inner padding.
---------	----------------

Example – to set up form content to have ten pixel padding:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialogbody {
  padding: 10px;
}
```

All static labels in the dialog box form are controlled with

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialoglabel
```

This class is not suitable for controlling the label size or position because you can apply it to texts in various places of the form user interface.

CSS properties of the dialog box label.

font-weight	Label font weight.
font-size	Label font size.
font-family	Label font family.
color	Label text color.

Dialog box labels can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up all labels to be gray, bold with a nine pixel font:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialoglabel {
  color: #666666;
  font-size: 9pt;
  font-weight: bold;
}
```

The size of the text copy displayed on top of the link is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialoginputwide
```

CSS properties of the dialog box input wide field

width	Text width.
padding	Inner padding.

Example – to set text copy to be 430 pixels wide and have a ten pixel padding in the bottom:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialoginputwide {
  padding-bottom: 10px;
  width: 430px;
}
```

The share link is wrapped in a container and controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

border	Border around the share link container.
padding	Inner padding.

Example – to set a one pixel grey border around embed code text and have nine pixels of padding:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialoginputcontainer {
  border: 1px solid #CCCCCC;
  padding: 9px;
}
```

The share link itself is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialoglink
```

CSS properties of the dialog box share link

width	Share link width.
-------	-------------------

Example – to set the share link to be 450 pixels wide:

```
.s7ecatalogsearchviewer .s7linkdialog .s7dialoglink {
  width: 450px;
}
```

Main control bar

The main control bar is the rectangular area on desktop systems and tablets that contain all user interface controls (except Large Page buttons) available for the eCatalog Search viewer.

On mobile phones, it still keeps Thumbnails, Table of Contents, Download, Print, Favorites, Social share, Full Screen, and Close buttons. However, First Page and Last Page buttons, and Page Indicator are removed from the main control bar and added to the secondary control bar instead. By default, the main control bar is displayed in the top of the viewer area on desktop systems and mobile phones, and moved to the bottom of the viewer area on tablets. It always takes the entire available viewer width. It is possible to change its color, height, and vertical position in the CSS, relative to the viewer container.

The appearance of the main control bar is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7controlbar
```

CSS property	Description
top	Position from the top of the viewer.
bottom	Position from the bottom of the viewer.
height	The height of the main control bar.
background-color	The background color of the main control bar.

Example – to set up a gray main control bar that is 36 pixels tall and is positioned at the top of the viewer container.

```
.s7ecatalogsearchviewer .s7controlbar {
  top: 0px;
  height: 36px;
  background-color: rgba(0, 0, 0, 0.5);
}
```

The main control bar supports an optional scroll feature. It is activated if the viewer width is too small and there is not enough space to fit all the buttons preset in the control bar. In this case, a two-state arrow button appears in the right-hand side of the control bar. Clicking or tapping on this button scrolls all the control bar elements to the left or to the right, depending on the scroll button state. The primary use case for this feature are mobile devices with small screens in portrait orientation.

The scroll feature is enabled for the main control bar, and is disabled for the secondary control bar. The feature is turned on and off using the following CSS class selector:

```
.s7ecatalogsearchviewer .s7controlbar .s7innercontrolbarcontainer
```

CSS property	Description
position	When set to <code>static</code> the scroll feature is disabled. Set this property to <code>absolute</code> to enable the scroll feature.

The scroll button is added to a special container element that positions the button properly and lets you style the area around the button differently from the rest of the control bar background in case the height of the scroll button is smaller than the control bar height.

The appearance of this scroll button container is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7controlbar .s7scrollbuttoncontainer
```

CSS property	Description
width	Normally should be equal or larger than the width of the scroll button itself.
background-color	Container background color.

You can size and skin the scroll button itself by way of CSS.

The appearance of this button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7controlbar .s7scrolllefttrightbutton
```

CSS property	Description
width	Width of button.
height	Height of button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` and `selected` attribute selectors, which can be used to apply different skins to different button states. In particular, `state="selected"` corresponds to the initial scroll button state when it is possible to scroll control bar contents to the left; `state="default"` corresponds to the state when the content is scrolled all the way to the left and the scroll button suggests to return it to the initial state.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to enable the scroll feature in the main control bar for mobile phones, and set up a scroll button that is 64 x 64 pixels that displays a different image for each of the 4 different button states when selected or not selected:

```
.s7ecatalogsearchviewer.s7size_small .s7controlbar .s7innercontrolbarcontainer {
  position: absolute;
}
.s7ecatalogsearchviewer.s7size_small.s7touchinput .s7controlbar .s7scrollbuttoncontainer {
  width:64px;
  background-color: rgb(0, 0, 0);
}
.s7ecatalogsearchviewer.s7size_small.s7touchinput .s7controlbar .s7scrolllefttrightbutton {
  width:64px;
  height:64px;
}
.s7ecatalogsearchviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='true'][state='up'] {
  background-image:url(images/v2/ControlBarLeftButton_dark_up_touch.png);
}
.s7ecatalogsearchviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='true'][state='over'] {
  background-image:url(images/v2/ControlBarLeftButton_dark_over_touch.png);
}
.s7ecatalogsearchviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='true'][state='down'] {
  background-image:url(images/v2/ControlBarLeftButton_dark_down_touch.png);
}
.s7ecatalogsearchviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='true'][state='disabled'] {
  background-image:url(images/v2/ControlBarLeftButton_dark_disabled_touch.png);
}
.s7ecatalogsearchviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='false'][state='up'] {
  background-image:url(images/v2/ControlBarRightButton_dark_up_touch.png);
}
```

```
.s7ecatalogsearchviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='false'][state='over'] {
  background-image:url(images/v2/ControlBarRightButton_dark_over_touch.png);
}
.s7ecatalogsearchviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='false'][state='down'] {
  background-image:url(images/v2/ControlBarRightButton_dark_down_touch.png);
}
.s7ecatalogsearchviewer.s7size_small.s7touchinput .s7controlbar
.s7scrolllefttrightbutton[selected='false'][state='disabled'] {
  background-image:url(images/v2/ControlBarRightButton_dark_disabled_touch.png);
}
```

Main viewer area

The main view area is the area occupied by the catalog image. It usually sets to fit the available device screen when no size is specified.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer
```

CSS property	Description
width	The width of the viewer.
height	The height of the viewer.
background-color	Background color in hexadecimal format.

Example – to set up a viewer with a white background (#FFFFFF) and make its size 512 x 288 pixels.

```
.s7ecatalogsearchviewer {
  background-color: #FFFFFF;
  width: 512px;
  height: 288px;
}
```

Next page button

Clicking or tapping on this button brings the user to the next page in the catalog. This button appears in the main control bar. This button is not displayed on mobile phones to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7toolbarrightbutton .s7panrightbutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.

CSS property	Description
left	Position from the left border of the main control bar, including padding.
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a next page button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 250 pixels from the right edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7toolbarrightbutton .s7panrightbutton {
  bottom:4px;
  right:250px;
  width:32px;
  height:32px;
}
.s7ecatalogsearchviewer .s7toolbarrightbutton .s7panrightbutton [state='up'] {
  background-image:url(images/v2/ToolBarRightButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7toolbarrightbutton .s7panrightbutton [state='over'] {
  background-image:url(images/v2/ToolBarRightButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7toolbarrightbutton .s7panrightbutton [state='down'] {
  background-image:url(images/v2/ToolBarRightButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7toolbarrightbutton .s7panrightbutton [state='disabled'] {
  background-image:url(images/v2/ToolBarRightButton_dark_disabled.png);
}
```

Page indicator

Page indicator displays current page index and total page count. It appears in main control bar on desktop systems and tablet, on mobile phones it is added to secondary control bar. Page indicator can be sized, skinned, and positioned by CSS.

The appearance page indicator is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7pageindicator
```


CSS property	Description
top	Position from the top border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
right	Position from the right border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
left	Position from the left border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
bottom	Position from the bottom border of the main control bar (on desktop systems and tablets) or secondary control bar (on mobile phones), including padding.
width	Width of the page indicator.
height	Height of the page indicator.
color	Font color.
font-family	Font name.
font-size	Font size.

Example – to set up a page indicator that is 56 x 28 pixels, horizontally centered and positioned 4 pixels from the bottom of the main control bar, and use a 14 pixel Helvetica font.

```
.s7ecatalogsearchviewer .s7pageindicator {
  position:absolute;
  bottom: 4px;
  margin-left: -28px;
  left: 50%;
  width:56px;
  height:28px;
  font-family: Helvetica;
  font-size:14px;
}
```

Page view

Main view consists of the catalog image. It can be swiped to get to another page or zoomed.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7pageview
```

CSS property	Description
background-color	Background color of the main view in hexadecimal format.

CSS property	Description
cursor	Cursor that is displayed over the main view.

Example – to make the main view transparent.

```
.s7ecatalogsearchviewer .s7pageview {
  background-color: transparent;
}
```

On desktop systems the component supports the `cursor` attribute selector which can be applied to `.s7pageview` class and controls the type of the cursor based on component state and user action. The following `cursor` values are supported:

Value	Description
default	Displayed when the image is not zoomable because of a small image resolution, component settings, or both.
zoomin	Displayed when the image can be zoomed in.
reset	Displayed when the image is at maximum zoom level and can be reset to initial state.
drag	Displayed when user pans the image which is in zoomed in state.
slide	Displayed when the user performs an image swap by doing horizontal swipe or flick.

The page divider that visually separates the left and right pages of the catalog spread is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7pageview .s7pagedivider
```

CSS property	Description
width	The width of page divider. Set to 0 px to hide the divider completely.
background-image	The image that you want to use as the page divider.

Example – to have 40 pixels wide page divider with semi-transparent image.

```
.s7ecatalogsearchviewer .s7pageview .s7pagedivider {
  width: 40px;
  background-image: url(images/sdk/divider.png);
}
```



Note: When the `frametransition` modifier is set to `turn` or `auto` (on desktop systems), the appearance of the page divider is controlled with the `pageturnstyle` modifier and the `.s7pagedivider` CSS class is ignored.

It is possible to configure the display of the custom mouse cursors over the main viewer area. This is controlled with the additional attribute selectors applied to `.s7ecatalogsearchviewer .s7pageview` CSS class:

CSS property	Description
default	Normally an arrow, displays for non-zoomable image.
zoomin	Shows when an image can be zoomed in.
reset	Shows when an image is at maximum zoom and can be reset.
drag	Shows when user performs drag operation on zoomed in image
slide	Shows when user performs image swap using slide gesture

Example – have different mouse cursors for each type of component state.

```
.s7ecatalogsearchviewer .s7pageview[cursortype="default"] {
  cursor:auto;
}
.s7ecatalogsearchviewer .s7pageview[cursortype="zoomin"] {
  cursor:url(images/zoomin_cursor.cur), auto;
}
.s7ecatalogsearchviewer .s7pageview[cursortype="reset"] {
  cursor:url(images/zoomout_cursor.cur), auto;
}
.s7ecatalogsearchviewer .s7pageview [cursortype="slide"] {
  cursor:url(images/slide_cursor.cur), auto;
}
.s7ecatalogsearchviewer .s7pageview[cursortype="drag"] {
  cursor:url(images/drag_cursor.cur), auto;
}
```

Previous page button

Clicking or tapping on this button brings the user to the previous page in the catalog. This button appears in the main control bar. This button is not displayed on mobile phones to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7toolbarleftbutton .s7panleftbutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.
left	Position from the left border of the main control bar, including padding.

CSS property	Description
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a previous page button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 250 pixels from the right edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7toolbarleftbutton .s7panleftbutton {
bottom:4px;
left:250px;
width:32px;
height:32px;
}
.s7ecatalogsearchviewer .s7toolbarleftbutton .s7panleftbutton [state='up'] {
background-image:url(images/v2/ToolBarLeftButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7toolbarleftbutton .s7panleftbutton [state='over'] {
background-image:url(images/v2/ToolBarLeftButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7toolbarleftbutton .s7panleftbutton [state='down'] {
background-image:url(images/v2/ToolBarLeftButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7toolbarleftbutton .s7panleftbutton [state='disabled'] {
background-image:url(images/v2/ToolBarLeftButton_dark_disabled.png);
}
```

Print

Print tool consists of a button added to the control bar and the modal dialog box that displays when the tool is activated.

The appearance of the print button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7print
```

CSS properties of the print button

margin-top	The offset from the top of the control bar.
margin-left	The distance to the next button on the left, or the left side of the control bar if this is the first button in a row.

width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – To set up a print button that is 28 x 28 pixels, and displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7print {
margin-top: 4px;
margin-left: 10px;
width: 28px;
height: 28px;
}
.s7ecatalogsearchviewer .s7print[state='up'] {
background-image: url(images/v2/Print_dark_up.png);
}
.s7ecatalogsearchviewer .s7print[state='over'] {
background-image: url(images/v2/Print_dark_over.png);
}
.s7ecatalogsearchviewer .s7print[state='down'] {
background-image: url(images/v2/Print_dark_down.png);
}
.s7ecatalogsearchviewer .s7print[state='disabled'] {
background-image: url(images/v2/Print_dark_disabled.png);
}
```

The background overlay which covers the web page when the dialog box is active is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7backoverlay
```

CSS properties of the back overlay

opacity	Background overlay opacity.
background-color	Background overlay color.

Example – to set up background overlay to be gray with 70% opacity:

```
.s7ecatalogsearchviewer .s7printdialog .s7backoverlay {
opacity: 0.7;
background-color: #222222;
}
```

By default the modal dialog is displayed centered in the screen on desktop systems. The positioning and sizing of the dialog box is managed by the component The dialog is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7kprintdialog .s7dialog
```

CSS properties of the dialog box

border-radius	Dialog box border radius.
background-color	Dialog box background color;

Example – to set up a dialog box to have a grey background:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialog {
background-color: #dddddd;
}
```

The dialog box header consists of an icon, a title text, and a close button. The header container is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogheader
```

CSS properties of the dialog box header

padding	Inner padding for header content.
---------	-----------------------------------

The icon and the title text are wrapped into an additional container controlled with the following:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogheader .s7dialogline
```

CSS properties of the dialog line

padding	Inner padding for the header icon and title.
---------	--

Header icon is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogheadericon
```

CSS properties of the dialog box header icon

width	Icon width.
height	Icon height.
background-image	Icon image.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .

Header title is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogheadertext
```

CSS properties of the dialog box header text

font-weight	Font weight.
-------------	--------------

font-size	Font height.
font-family	Font family.
padding	Internal text padding.

Close button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7closebutton
```

CSS properties of the close button

top	Vertical button position relative to header container.
right	Horizontal button position relative to header container.
width	Button width.
height	Button height.
padding	Inner padding of button.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The Close button tool tip and the dialog box title can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up dialog header with padding, 22 x 22 pixels icon, bold 16 point title, and a 28 x 28 pixel Close button positioned two pixels from the top and two pixels from the right of dialog box container:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogheader {
  padding: 10px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogheader .s7dialogline {
  padding: 10px 10px 2px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogheadericon {
  background-image: url("images/sdk/dlgprint_cap.png");
  height: 22px;
  width: 22px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogheadertext {
  font-size: 16pt;
  font-weight: bold;
  padding-left: 16px;
}
.s7ecatalogsearchviewer .s7printdialog .s7closebutton {
  top: 2px;
  right: 2px;
```

```
padding:8px;
width:28px;
height:28px;
}
.s7ecatalogsearchviewer .s7printdialog .s7closebutton[state='up'] {
background-image:url(images/sdk/close_up.png);
}
.s7ecatalogsearchviewer .s7printdialog .s7closebutton[state='over'] {
background-image:url(images/sdk/close_over.png);
}
.s7ecatalogsearchviewer .s7printdialog .s7closebutton[state='down'] {
background-image:url(images/sdk/close_down.png);
}
.s7ecatalogsearchviewer .s7printdialog .s7closebutton[state='disabled'] {
background-image:url(images/sdk/close_disabled.png);
}
```

Dialog box footer consists of Cancel and Send to Print buttons. The footer container is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogfooter
```

CSS properties of the dialog box footer

border	Border that you can use to visually separate the footer from the rest of the dialog box.
--------	--

The footer has an inner container that keeps both buttons. It is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogbuttoncontainer
```

CSS properties of the dialog box button container


padding	Inner padding between the footer and the buttons.
---------	---

Cancel button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogcancelbutton
```

CSS properties of the dialog box cancel button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.

 **Note:** This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

Send to Print button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogactionbutton
```

CSS properties of the dialog box action button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

In addition, both buttons share the same common CSS class which can contain CSS settings that are the same for other dialog box buttons:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogfooter .s7button
```

CSS properties of the button

font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.
line-height	Text height inside the button. Affects vertical alignment.
box-shadow	Drop shadow.
margin-right	Right button margin.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box footer with 64 x 34 Cancel button and a 96 x 34 Send to Print button, with the text color and background color different for each button state:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogfooter {
  border-top: 1px solid #909090;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogbuttoncontainer {
  padding-bottom: 6px;
  padding-top: 10px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogfooter .s7button {
  box-shadow: 1px 1px 1px #999999;
  color: #FFFFFF;
  font-size: 9pt;
  font-weight: bold;
  line-height: 34px;
  margin-right: 10px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogcancelbutton {
  width: 64px;
  height: 34px;
}
```

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogcancelbutton[state='up'] {
  background-color:#666666;
  color:#dddddd;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogcancelbutton[state='down'] {
  background-color:#555555;
  color:#ffffff;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogcancelbutton[state='over'] {
  background-color:#555555;
  color:#ffffff;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogcancelbutton[state='disabled'] {
  background-color:#b2b2b2;
  color:#dddddd;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogactionbutton {
  width:96px;
  height:34px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogactionbutton[state='up'] {
  background-color:#333333;
  color:#dddddd;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogactionbutton[state='down'] {
  background-color:#222222;
  color:#cccccc;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogactionbutton[state='over'] {
  background-color:#222222;
  color:#cccccc;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogactionbutton[state='disabled'] {
  background-color:#b2b2b2;
  color:#dddddd;
}
```

The main dialog area (between the header and the footer) contains dialog content. In all cases, the component manages the width of this area, it is not possible to set it in CSS. The main dialog area is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogviewarea
```

CSS properties of the dialog box viewing area

height	The height of the main dialog box area.
background-color	The background color of the main dialog box area.
margin	Outer margin.

Example – to set up a main dialog area to have an automatically calculated height, have a ten pixel margin, and use a white background:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogviewarea {
  background-color:#ffffff;
  margin:10px;
  height:auto;
}
```

All form content (like labels and input fields) resides inside a container controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogbody
```

CSS properties of the dialog box body

padding	Inner padding.
---------	----------------

Example – to set up form content to have ten pixel padding:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogbody {
  padding: 10px;
}
```

Dialog box form is filled on line-by-line basis, where each line carries a part of the form content (like a label and a text input field). Single form line is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogbody .s7dialogline
```

CSS properties of the dialog box line

padding	Inner line padding.
---------	---------------------

Example – to set up a dialog box form to have ten pixel padding for each line:

```
.s7ecatalogsearchviewer .s7emaildialog .s7dialogbody .s7dialogline {
  padding: 10px;
}
```

The size of the block of dialog content is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialoginputwide
```

CSS properties of the dialog input width

width	Block width.
padding	Inner line padding.

Example – to set a content block to be 430 pixels wide and have 10 pixels padding in the bottom:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialoginputwide {
  padding-bottom: 10px;
  width: 430px;
}
```

All static labels in the dialog box form are controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialoglabel
```

This class is not suitable for controlling the label size or position because you can apply it to texts in various places in the form user interface.

CSS properties of the dialog box label.

font-weight	Label font weight.
font-size	Label font size.
font-family	Label font family.
color	Label text color.

Dialog box labels can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up all labels to be gray, bold, with a nine pixel font:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialoglabel {
  color: #666666;
  font-size: 9pt;
  font-weight: bold;
}
```

Input controls are wrapped into the container and controlled with the following CSS class selector::

```
.s7ecatalogsearchviewer .s7printdialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

padding-left	Inner padding.
--------------	----------------

Example – to set a 30 pixel padding from the left edge of the dialog box.

```
.s7ecatalogsearchviewer .s7printdialog .s7dialoginputcontainer {
  padding-left: 30px;
}
```

Radio buttons and their caption text are controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogoption
```

CSS properties of the dialog box option

width	The total width of the radio button with a caption.
color	Caption text color.

The spacing between the radio button and its caption is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogoptioninput
```

CSS properties of the dialog box option input

margin-right	Spacing between the radio button and its caption.
--------------	---

Numeric pickers for print range selection are controlled with the following CSS class selector

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogrange
```

CSS properties of the dialog box print range

width	Width of the numeric picker.
margin	Spacing around the numeric picker.

Example – to set up all radio buttons to be 150 pixels wide with black text, ten pixel spacing, and 42 pixel wide numeric pickers:

```
.s7ecatalogsearchviewer .s7printdialog .s7dialogoption {
  color: #000000;
  width: 150px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogoption input {
```

```
margin-right: 10px;
}
.s7ecatalogsearchviewer .s7printdialog .s7dialogrange {
margin-left: 10px;
margin-right: 10px;
width: 42px;
}
```

The horizontal divider between the page range selection and the print layout sections is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer
.s7printdialog .s7horizontaldivider
```

CSS properties of the horizontal divider

border	Border around divider.
padding	Inner padding.
width	Divider width.
margin	Outer margin

Example – to set up a 430 pixel wide grey divider with a 10 pixel vertical padding on both sides, and a ten pixel margin at the top:

```
.s7ecatalogsearchviewer .s7printdialog .s7horizontaldivider {
border-top: 1px solid #aaaaaa;
margin-top: 10px;
padding-bottom: 10px;
padding-top: 10px;
width: 430px;
}
```

Remove Favorite button

The position of the Remove Favorite button is fully managed by the Favorites menu.

The appearance of the Remove Favorite button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7removefavoritebutton
```

CSS properties of the Remove Favorite button

background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
width	Width of the button.
height	Height of the button.



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the state when a user can add a new Favorite icon by clicking or tapping. *selected='false'* corresponds to the normal operation mode when a user can zoom, pan, and swap pages.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Remove Favorite button that is 28 x 28 pixels, and displays a different image for each of the four different button states when selected or not selected.

```
.s7ecatalogsearchviewer .s7removefavoritebutton {
  width:28px;
  height:28px;
}
.s7ecatalogsearchviewer .s7removefavoritebutton[selected='false'][state='up'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7removefavoritebutton[selected='false'][state='over'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7removefavoritebutton[selected='false'][state='down'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7removefavoritebutton[selected='false'][state='disabled'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_disabled.png);
}
.s7ecatalogsearchviewer .s7removefavoritebutton[selected='true'][state='up'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7removefavoritebutton[selected='true'][state='over'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7removefavoritebutton[selected='true'][state='down'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7removefavoritebutton[selected='true'][state='disabled'] {
background-image:url(images/v2/RemoveFavoriteButton_dark_disabled.png);
}
```

Search button

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7searchbutton
```

CSS property	Description
width	Width of the button.
height	Height of the button.
margin-top	Te offset from the top of the control bar.
margin-left	The distance to the next button on the left, or the left side of the control bar if this is the first button in a row.

CSS property	Description
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` and `selected` attribute selectors, which can be used to apply different skins to different button states.

In particular, `selected='false'` corresponds to the initial scroll button state, and `selected='true'` corresponds to the state when the search panel is active.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Search button that is 28 x 28 pixels, and displays a different image for each of the four different button states when selected or not selected.

```
.s7ecatalogsearchviewer .s7searchbutton{
  margin-top: 4px;
  margin-left: 10px;
  width:28px;
  height:28px;
  display: inline-block;
  background-size:contain;
}

.s7ecatalogsearchviewer.s7mouseinput .s7searchbutton[selected='false'][state='up'] {
  background-image:url(images/v2/Search_dark_up.png);
}
.s7ecatalogsearchviewer.s7mouseinput .s7searchbutton[selected='false'][state='over'] {
  background-image:url(images/v2/Search_dark_over.png);
}
.s7ecatalogsearchviewer.s7mouseinput .s7searchbutton[selected='false'][state='down'] {
  background-image:url(images/v2/Search_dark_down.png);
}
.s7ecatalogsearchviewer.s7mouseinput .s7searchbutton[selected='false'][state='disabled'] {
  background-image:url(images/v2/Search_dark_disabled.png);
}
.s7ecatalogsearchviewer.s7mouseinput .s7searchbutton[selected='true'][state='up'] {
  background-image:url(images/v2/Search_dark_over.png);
}
.s7ecatalogsearchviewer.s7mouseinput .s7searchbutton[selected='true'][state='over'] {
  background-image:url(images/v2/Search_dark_over.png);
}
.s7ecatalogsearchviewer.s7mouseinput .s7searchbutton[selected='true'][state='down'] {
  background-image:url(images/v2/Search_dark_over.png);
}
.s7ecatalogsearchviewer.s7mouseinput .s7searchbutton[selected='true'][state='disabled'] {
  background-image:url(images/v2/Search_dark_disabled.png);
}
```

Search effect

The viewer displays search result regions over the main view to highlight words or phrases found in the catalog.

CSS properties of the main viewer area

The appearance of search result regions is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7searcheffect .s7region
```

CSS property	Description
background	Background of search result region.

Example – to set up search result regions with a semi-transparent, yellow fill:

```
.s7ecatalogsearchviewer .s7searcheffect .s7region {
  background: rgba(255,255,0, 0.5);
}
```

Search results panel

The search results panel consists of the search input box at the top and the main area where informational messages or search results are displayed.

CSS properties of the main viewer area

When the panel is active the viewer user interface is covered with a semi-transparent fill. The color and opacity of this fill is controlled with the following CSS class selector:

```
.s7ecatalogviewer .s7searchpanel .s7backoverlay
```

CSS property	Description
background-color	Color of the overlay.
opacity	Opacity of the color.

The search results panel always occupies all available viewer height. However, you can configure the width. You can set the width to an absolute pixel value, which is a default setting for medium and large size breakpoints. Or, you can set the width to 100% to make the search results panel occupy the entire viewer area. The panel width is controlled by the following CSS class selector:

```
.s7ecatalogsearchviewer .s7searchpanel .s7searchresultspace
```

CSS property of the search result space

width	Width of the search result space.
-------	-----------------------------------

Example – to set up a 250 pixel wide search results panel on large and medium size breakpoints and use a full-size panel on a small size breakpoint:

```
.s7ecatalogsearchviewer.s7size_large .s7searchpanel .s7searchresultspace,
.s7ecatalogsearchviewer.s7size_medium .s7searchpanel .s7searchresultspace {
  width:250px;
}
.s7ecatalogsearchviewer.s7size_small .s7searchpanel .s7searchresultspace {
  width:100%;
}
```


The top of the search results panel is dedicated to the search input box. The padding on the sides of the input box is controlled by the following CSS class selector:

```
.s7ecatalogsearchviewer .s7searchpanel .s7searchinputcontainer
```

CSS properties of the search input container

padding	Padding around the input box.
---------	-------------------------------

The search input field is controlled by the following CSS class selector:

```
.s7ecatalogsearchviewer .s7searchpanel .s7searchinput
```

CSS properties of the search input field

width	Width of search input field.
height	Height of search input field.
background-image	The URL to the "looking glass" icon image shown on the left side of the input field;
background-size	The size of the "looking glass" icon.
padding-left	The inner padding between the input field bounds and the input text. Normally, this value is set to the same value as the width of the icon.
border	Border of the search input field.
margin	Margin of the search input field
font-size	Size of the text font.

Example – to set up a search input field with a 30 x 30 pixel "looking glass" icon, 30 pixels height, and 14 pixels text font:

```
.s7ecatalogsearchviewer .s7searchpanel .s7searchinput {
padding-left:30px;
height:30px;
background-size:30px 30px;
background-image: url(images/v2/Search_form_field.png);
font-size:14px;
}
```

The search results panel may display a textual prompt when the feature is first called. It also shows the user a message when their search did not return any results. In all cases, text appears in the main part of the search results panel and is controlled by the following CSS class selector:

```
.s7ecatalogsearchviewer .s7searchpanel .s7searchinfo
```

CSS properties of the search information

color	Color of text.
font-family	Name of text font.

font-align	Horizontal text alignment.
font-size	Size of font text.



Note: This text panel supports the *state* attribute selector, which can be used to apply different styles to different text messages. In particular, *state='prompt'* corresponds to the text prompt shown when the panel is called for the first time; *state='results'* corresponds to the text with information about search hits; and *state='no_results'* corresponds to the text shown when the search query did not return any results.

The message text can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a text panel that uses a gray 18 pixel font:

```
.s7ecatalogsearchviewer .s7searchpanel .s7searchinfo {
  font-size: 18px;
  color: #696969;
}
```

Search results are rendered as a single column or single row of thumbnails for pages with search hits. The spacing between search results thumbnails is controlled with the following CSS class selector:

```
.ecatalogsearchviewer .s7searchpanel .s7swatches .s7thumbcell
```

CSS properties of the thumbnail cells

margin	The size of the vertical margin around each thumbnail. Actual thumbnail spacing equals the sum of the top and bottom margins set for <code>.s7thumbcell</code> .
--------	--

Example – to set up 10 pixel spacing:

```
.s7ecatalogsearchviewer .s7searchpanel .s7swatches .s7thumbcell {
  margin: 5px;
}
```

The appearance of individual thumbnails is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7searchpanel .s7swatches .s7thumb
```

CSS properties of the thumbnail

width	Width of the thumbnail.
height	Height of the thumbnail.
border	Border of the thumbnail.

Example – to set up thumbnails that are 215 x 129 pixels, have a light grey default border, and a dark grey selected border:

```
.s7ecatalogsearchviewer .s7searchpanel .s7swatches .s7thumb {
  width: 215px;
  height: 129px;
}
```

The appearance of the thumbnail label is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer  
.s7searchpanel .s7swatches .s7label
```

CSS properties of the label

color	Text color.
font-family	Name of text font.
font-size	Size of text font.

Example – to set up labels that use 12 pixel, grey, Helvetica font:

```
.s7ecatalogsearchviewer .s7searchpanel .s7swatches .s7label {  
  font-family: Helvetica,sans-serif;  
  color: #4c4c4c;  
  font-size: 12px;  
}
```

On systems that use mouse input, two scroll buttons appear at the bottom of the search results panel to a user scroll through the search results. The appearance of the up and down scroll buttons are controlled with following CSS class selectors:

```
.s7ecatalogsearchviewer .s7searchpanel .s7scrollupbutton  
.s7ecatalogsearchviewer .s7searchpanel .s7scrolldownbutton
```

It is not possible to position scroll buttons using CSS top, left, bottom, and right properties. Instead, the viewer logic positions them automatically.

CSS properties of the scroll up and down buttons

width	Width of the scroll button.
height	Height of the scroll button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note:

This button supports the `state` attribute selector, which can be used to apply different skins to "up", "down", "over", and "disabled" button states.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a scroll up button that is 125 x 35 pixels and has different artwork for each state:

```
.s7ecatalogsearchviewer .s7searchpanel .s7scrollupbutton {  
  width:125px;  
  height:35px;  
}  
.s7ecatalogsearchviewer .s7searchpanel .s7scrollupbutton[state='up'] {
```

```

background-image:url(images/sdk/searchpanel_scroll_up_up.png);
}
.s7ecatalogsearchviewer .s7searchpanel .s7scrollupbutton[state='over'] {
background-image:url(images/sdk/searchpanel_scroll_up_over.png);
}
.s7ecatalogsearchviewer .s7searchpanel .s7scrollupbutton[state='down'] {
background-image:url(images/sdk/searchpanel_scroll_up_down.png);
}
.s7ecatalogsearchviewer .s7searchpanel .s7scrollupbutton[state='disabled'] {
background-image:url(images/sdk/searchpanel_scroll_up_disabled.png);
}
.s7ecatalogsearchviewer .s7searchpanel .s7scrolldownbutton {
width:125px;
height:35px;
}
.s7ecatalogsearchviewer .s7searchpanel .s7scrolldownbutton[state='up'] {
background-image:url(images/sdk/searchpanel_scroll_down_up.png);
}
.s7ecatalogsearchviewer .s7searchpanel .s7scrolldownbutton[state='over'] {
background-image:url(images/sdk/searchpanel_scroll_down_over.png);
}
.s7ecatalogsearchviewer .s7searchpanel .s7scrolldownbutton[state='down'] {
background-image:url(images/sdk/searchpanel_scroll_down_down.png);
}
.s7ecatalogsearchviewer .s7searchpanel .s7scrolldownbutton[state='disabled'] {
background-image:url(images/sdk/searchpanel_scroll_down_disabled.png);
}

```

Secondary control bar

The secondary control bar is the rectangular area that contains First and Last Page buttons and a Page Indicator when made available in CSS.

By default, it is displayed on mobile phones only and is located in the bottom of the viewer. It always takes the whole available viewer width. It is possible to change its color, height and vertical position by CSS, relative to the viewer container.

The appearance of the secondary control bar is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7secondarycontrols .s7controlbar
```

CSS property	Description
top	Position from the top of the viewer.
bottom	Position from the bottom of the viewer.
height	The height of the main control bar.
background-color	The background color of the secondary control bar.

Example – to set up a gray secondary control bar that is 72 pixels tall and is positioned at the bottom of the viewer container.

```

.s7ecatalogsearchviewer .s7secondarycontrols .s7controlbar {
bottom: 0px;
height: 72px;
}

```

Social share

The social share tool appears in the top left corner by default. It consists of a button and a panel which expands when user clicks or taps on a button and contains individual sharing tools.

The position and size of the social share tool in the viewer user interface is controlled with the following:

```
.s7ecatalogsearchviewer .s7socialshare
```

CSS properties of the social share tool

margin-top	The offset from the top of the control bar.
margin-left	The distance to the next button on the left, or the left side of the control bar if this is the first button in a row.
width	The width of the social sharing tool.
height	The height of the social sharing tool.

Example – set up a social sharing tool that is positioned four pixels from the top and five pixels from the right of viewer container and is sized to 28 x 28 pixels.

```
.s7ecatalogsearchviewer .s7socialshare {
margin-top: 4px;
margin-left: 10px; width:28px;
height:28px;
}
```

The appearance of the social share tool button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7socialshare .s7socialbutton
```

CSS properties of the social button

background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – set up a social sharing tool button that displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7socialshare .s7socialbutton[state='up'] {
background-image:url(images/v2/SocialShare_video_dark_up.png);
}
.s7ecatalogsearchviewer .s7socialshare .s7socialbutton[state='over'] {
background-image:url(images/v2/SocialShare_dark_over.png);
}
.s7ecatalogsearchviewer .s7socialshare .s7socialbutton[state='down'] {
background-image:url(images/v2/SocialShare_dark_down.png);
}
.s7ecatalogsearchviewer .s7socialshare .s7socialbutton[state='disabled'] {
```

```
background-image:url(images/v2/SocialShare_dark_disabled.png);
}
```

The appearance of the panel which contains the individual social sharing icons is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7socialshare .s7socialsharepanel
```

CSS properties of the social share panel

background-color	The background color of the panel.
------------------	------------------------------------

Example – set up a panel to have transparent color:

```
.s7ecatalogsearchviewer .s7socialshare .s7socialsharepanel {
  background-color: transparent;
}
```

Table of contents

Table of contents is a button located in the main control bar. When activated, a drop-down panel appears with a list of page indexes and labels.

Based on the configuration, the list can contain all pages that are present in the catalog or only those pages that have explicit labels defined. On desktop systems, if the list is longer than the available screen real estate, a scroll bar is displayed on the right.

The position and size of the table of contents button in the viewer user interface is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7tableofcontents
```

CSS properties of the table of contents

margin-top	The offset from the top of the control bar.
margin-left	The distance to the next button on the left, or the left side of the control bar if this is the first button in a row.
width	The width of the table of contents button.
height	The height of the table of contents button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – set up a table of contents button that is positioned 4 pixels from the bottom and 43 pixels from the left of the main control bar; size is 28 x 28 pixels and a different image is displayed for each of the four different button states:

```
.s7ecatalogsearchviewer .s7tableofcontents {
margin-top: 4px;
margin-left: 10px; width: 28px;
height: 28px;
}
.s7ecatalogsearchviewer .s7tableofcontents[state='up'] {
background-image:url(images/v2/TableOfContents_dark_up.png);
}
.s7ecatalogsearchviewer .s7tableofcontents[state='over'] {
background-image:url(images/v2/TableOfContents_dark_over.png);
}
.s7ecatalogsearchviewer .s7tableofcontents[state='down'] {
background-image:url(images/v2/TableOfContents_dark_down.png);
}
.s7ecatalogsearchviewer .s7tableofcontents[state='disabled'] {
background-image:url(images/v2/TableOfContents_dark_disabled.png);
}
```

The appearance of the drop-down panel is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7panel
```

CSS properties of the drop-down panel

background-color	Background color of drop-down panel.
margin	Internal offset between the panel boundaries and the content.
box-shadow	Drop-shadow around the panel.



Note: It is not possible to control the size or the position of the drop-down panel from CSS; the component manages its layout programmatically.

Example – set up a drop-down panel that has a semi-transparent black background, a 5 pixel margin around the content, and a drop shadow:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7panel {
background-color: rgba(0, 0, 0, 0.5);
margin: 5px;
box-shadow: 2px 2px 3px #c0c0c0;
}
```

The individual item look and feel is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7panel .s7item
```

CSS properties of the item

font-family	Font name.
font-size	Font size.
height	Item's height.

padding	Internal padding.
---------	-------------------



Note: Drop-down list item supports the `state` attribute selector, which can be used to apply different skins to hover and selected item states.

Example – set up a drop-down item with a Helvetica 14 pixel font and 19 pixel high. An item has a dark gray background on hover and a light gray background when selected:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7panel .s7item {
font-family: Helvetica,sans-serif;
font-size: 14px;
height: 19px;
}
.s7ecatalogsearchviewer .s7tableofcontents .s7panel .s7item[state="over"] {
background-color: rgb(102, 102, 102);
}
.s7ecatalogsearchviewer .s7tableofcontents .s7panel .s7item[state="selected"] {
background-color: rgb(178, 178, 178);
}
```

An element that shows the page index is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7panel .s7index
```

CSS properties of the page index

min-width	Minimum element width.
max-width	Maximum element width.
padding-right	Distance between the page index and the page label.



Note: It is possible to hide the page index entirely by setting `display: none` for the `s7index` CSS class.

Example 1 – set up a page index with a minimum width of 40 pixels, a maximum width of 70 pixels, and a 5 pixel margin on the right-hand side:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7panel .s7index {
max-width: 70px;
min-width: 40px;
padding-right: 5px;
}
```

Example 2 – hide page index:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7panel .s7index {
display: none;
}
```

The page label is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7panel .s7label
```

CSS properties of the page label

min-width	Minimum element width.
-----------	------------------------

max-width	Maximum element width.
-----------	------------------------

Example – set up a page index with a minimum width of 40 pixels and a maximum width of 240 pixels:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7panel .s7label {
min-width: 40px;
max-width: 240px;
}
```

In case there are more items than can fit vertically within the drop-down panel and the system is a desktop, the component renders a vertical scroll bar on the right side of the panel. The appearance of the scroll bar area is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar
```

CSS properties of the scrollbar

width	The scroll bar width.
top	The vertical scroll bar offset from the top of the panel area.
bottom	The vertical scroll bar offset from the bottom of the panel area.
right	The horizontal scroll bar offset from the right edge of the panel area.

Example – set up a scroll bar that is 28 pixels wide and does not have a margin for the top, right, or bottom area of the panel:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar {
top:0px;
bottom:0px;
right:0px;
width:28px;
}
```

Scroll bar track is the area between the top and bottom scroll buttons. The component automatically sets the position and height of the track. The track is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrolltrack
```

CSS properties of the scroll track

width	The track width.
background-color	The track background color.

Example – set up a scroll bar track that is 28 pixels wide and has a semi-transparent gray background:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrolltrack {
width:28px;
background-color:rgba(102, 102, 102, 0.5);
}
```

The scroll bar thumb moves vertically within the scroll track area. Its vertical position is controlled by the component logic. However, thumb height does not dynamically change depending on the amount of content. You can configure the thumb height and other aspects with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrollthumb
```

CSS properties of the scrollbar thumb

width	The thumb width.
height	The thumb height.
padding-top	The vertical padding between the top of the track.
padding-bottom	The vertical padding between the bottom of the track.
background-image	The image that is displayed for a given thumb state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note:

Thumb supports the `state` attribute selector, which can be used to apply different skins to the up, down, over, and disabled thumb states.

Example – set up a scroll bar thumb that is 28 x 45 pixels, has 10 pixel margins on the top and bottom, and has different artwork for each state:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrollthumb {
  width:28px;
  background-repeat:no-repeat;
  background-position:center;
  height:45px;
  padding-top:10px;
  padding-bottom:10px;
}
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrollthumb[state='up'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_up.png);
}
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrollthumb[state='down'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_down.png);
}
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrollthumb[state='over'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_over.png);
}
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrollthumb[state='disabled'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_up.png);
}
```

The appearance of the top and bottom scroll buttons is controlled with the following CSS class selectors:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrollupbutton
```

```
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrolldownbutton
```

It is not possible to position the scroll buttons using CSS `top`, `left`, `bottom`, and `right` properties; instead, the viewer logic positions them automatically.

CSS properties of the scroll up and scroll down button

width	The button width.
height	The button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note:

Button supports the `state` attribute selector, which can be used to apply different skins to the up, down, over, and disabled button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – set up scroll buttons that are 28 x 32 pixels and have different artwork for each state:

```
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrollupbutton {
  width:28px;
  height:32px;
}
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrollupbutton[state='up'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrollupbutton[state='over'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrollupbutton[state='down'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrollupbutton[state='disabled'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrolldownbutton {
  width:28px;
  height:32px;
}
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrolldownbutton[state='up'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrolldownbutton[state='over'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrolldownbutton[state='down'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7tableofcontents .s7scrollbar .s7scrolldownbutton[state='disabled'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_up.png);
}
```

Thumbnails

Thumbnails consist of a grid of thumbnail images with an optional scroll bar on the right side to allow vertical scrolling.

Thumbnails are toggled by clicking the thumbnail button in the main control bar. When thumbnails are active, they display in modal mode overlaid on top of the viewer user interface. The viewer logic automatically resizes the thumbnails container to the entire viewer area.

The appearance of the thumbnails container is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7thumbnailgridview
```

CSS property	Description
top	The vertical offset of the thumbnails container from the top of the viewer.
margin-top	The top margin.
margin-left	The left margin.
margin-right	The right margin.
margin-bottom	The bottom margin.
background-color	The background color of the thumbnails area.

Example – to set up thumbnails to have 32 pixels offset from the top, 5 pixels margins on the left and right, and 8 pixels margin on the bottom, with 0xDDDDDD background.

```
.s7ecatalogsearchviewer .s7thumbnailgridview {
  top: 32px;
  margin-left: 5px;
  margin-right: 5px;
  margin-bottom: 8px;
  background-color: rgb(221, 221, 221);
}
```

The spacing between thumbnails is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7thumbcell
```

CSS property	Description
margin	The size of the horizontal and vertical margin around each thumbnail. Actual horizontal thumbnail spacing is equal to the sum of the left and right margin that is set for <code>.s7thumbcell</code> . Vertical thumbnail spacing equals to the sum of the top and bottom margin.

Example – to set a 10 pixel space both vertically and horizontally.

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7thumbcell {
  margin: 5px;
}
```

The appearance of individual thumbnail is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7thumb
```

CSS property	Description
width	The width of the thumbnail.
height	The height of the thumbnail.
border	The border of the thumbnail.
background-color	The background color of the thumbnail.

On touch devices, when rotated to portrait mode, the viewer may size thumbnails to half of what is configured in case it decides to split the catalog spread into individual pages.



Note: Thumbnail supports the *state* attribute selector, which can be used to apply different skins to different thumbnail states. In particular, *state="selected"* corresponds to the thumbnail for the image that is currently displayed in the main view, *state="default"* corresponds to the rest of thumbnails, and *state="over"* is used on mouse hover.

Example – to set up thumbnails that are 120 x 85 pixels, have a white background, a light gray standard border, and a dark grey selected border.

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7thumb {
  width:120px;
  height:85px;
  background-color: rgb(255, 255, 255);
  border: solid 1px #999999;
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7thumb[state="selected"]{
  border: solid 2px #666666;
}
```

The appearance of the thumbnail label is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7label
```

CSS property	Description
font-family	Font name.
font-size	Font size.

Example – to set up labels to use 14 pixels Helvetica font.

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7label {
  font-family: Helvetica,sans-serif;
  font-size: 12px;
}
```

In case there are more thumbnails than can fit vertically into the view, thumbnails renders the vertical scroll bar on the right side. The appearance of scroll bar area is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar
```

CSS property	Description
width	The width of the scroll bar.
top	The vertical scroll bar offset from the top of the thumbnails area.
bottom	The vertical scroll bar offset from the bottom of the thumbnails area.
right	The horizontal scroll bar offset from the right edge of the thumbnails area.

Example – to set up a scroll bar that is 28 pixels wide and has an 8 pixel margin from the top, right, and bottom of the thumbnails area.

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar {
  top:8px;
  bottom:8px;
  right:8px;
  width:28px;
}
```

The scroll bar track is the area between the top and bottom scroll buttons. The component automatically sets the position and height of the track. The track is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrolltrack
```

CSS property	Description
width	The width of the scroll bar track.
background-color	The background color of the scroll bar track.

Example – to set up a scroll bar track that is 28 pixels wide and has a semi-transparent gray background.

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrolltrack {
  width:28px;
  background-color:rgba(102, 102, 102, 0.5);
}
```

The scroll bar thumb moves vertically within the scroll track area. Its vertical position is fully controlled by the component logic, however, thumb height does not dynamically change depending on the amount of content. The thumb height and other aspects are controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrollthumb
```

CSS property	Description
width	The width of the scroll bar thumb.
height	The height of the scroll bar thumbnail.

CSS property	Description
padding-top	The vertical padding between the top of the scroll bar track.
padding-bottom	The vertical padding between the bottom of the scroll bar track.
background-image	The image that is displayed for a given thumb state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: Thumb supports the *state* attribute selector, which can be used to apply different skins to the thumb states *up*, *down*, *over*, and *disabled*.

Example – to set up a scroll bar thumb that is 28 x 45 pixels, has 10 pixel margins on the top and bottom, and have different artwork for each state.

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrollthumb {
  width:28px;
  background-repeat:no-repeat;
  background-position:center;
  height:45px;
  padding-top:10px;
  padding-bottom:10px;
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrollthumb[state='up'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_up.png);
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrollthumb[state='down'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_down.png);
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrollthumb[state='over'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_over.png);
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrollthumb[state='disabled'] {
  background-image:url(images/v2/ThumbnailScrollThumb_dark_up.png);
}
```

The appearance of the top and bottom scroll buttons is controlled with following CSS class selectors:

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrollupbutton
```

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrolldownbutton
```

It is not possible to position the scroll buttons using CSS *top*, *left*, *bottom*, and *right* properties. Instead, the viewer logic positions them automatically.

CSS property	Description
width	The width of the button.
height	The height of the button.

CSS property	Description
background-image	The image that is displayed for a given thumb state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: These buttons support the *state* attribute selector, which can be used to apply different skins to the different button states *up*, *down*, *over*, and *disabled*.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up scroll buttons which are 28 x 32 pixels and have different artwork for each state.

```
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrollupbutton {
  width:28px;
  height:32px;
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrollupbutton[state='up'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrollupbutton[state='over'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrollupbutton[state='down'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrollupbutton[state='disabled'] {
  background-image:url(images/v2/ThumbnailScrollUpButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrolldownbutton {
  width:28px;
  height:32px;
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrolldownbutton[state='up'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrolldownbutton[state='over'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrolldownbutton[state='down'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7thumbnailgridview .s7scrollbar .s7scrolldownbutton[state='disabled'] {
  background-image:url(images/v2/ThumbnailScrollDownButton_dark_up.png);
}
```

Thumbnails button

Clicking or tapping this button resets toggles the viewer between the main view and the thumbnails. This button appears in the main control bar. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7thumbnailpagebutton
```


CSS property	Description
margin-top	The offset from the top of the control bar.
margin-left	The distance to the next button on the left, or the left side of the control bar if this is the first button in a row.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the viewer state when thumbnail mode is active state and *selected='false'* corresponds to the default state with main view.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up thumbnails button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 5 pixels from the left edge of the main control bar, and displays a different image for each of the four different button states when selected or not selected.

```
.s7ecatalogsearchviewer .s7thumbnailpagebutton{
margin-top: 4px;
margin-left: 5px; width:28px;
height:28px;
}
.s7ecatalogsearchviewer .s7thumbnailpagebutton [selected='false'][state='up'] {
background-image:url(images/v2/ThumbnailPageButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7thumbnailpagebutton [selected='false'][state='over'] {
background-image:url(images/v2/ThumbnailPageButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7thumbnailpagebutton [selected='false'][state='down'] {
background-image:url(images/v2/ThumbnailPageButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7thumbnailpagebutton [selected='false'][state='disabled'] {
background-image:url(images/v2/ThumbnailPageButton_dark_disabled.png);
}
.s7ecatalogsearchviewer .s7thumbnailpagebutton [selected='true'][state='up'] {
background-image:url(images/v2/ThumbnailPageButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7thumbnailpagebutton [selected='true'][state='over'] {
background-image:url(images/v2/ThumbnailPageButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7thumbnailpagebutton [selected='true'][state='down'] {
background-image:url(images/v2/ThumbnailPageButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7thumbnailpagebutton [selected='true'][state='disabled'] {
background-image:url(images/v2/ThumbnailPageButton_dark_disabled.png);
}
```

Tooltips

On desktop systems some user interface elements like buttons have tool tips that are displayed on mouse hover.

CSS properties of the main viewer area

The appearance of tool tips is controlled with the following CSS class selector:

```
.s7tooltip
```

CSS property	Description
border-radius	Background border radius.
border-color	Background border color.
background-color	Background color.
color	Text color.
font-family	Text font name.
font-size	Text font size.



Note: In case tool tip styles are customized from within the embedding web page, all properties have to contain ! IMPORTANT rule. This is not necessary if tool tips are customized in the viewer's CSS file.

Example – to set up tool tips that have a gray border with 3px corner radius, black background and white text written with Arial, 11 pixels size:

```
.s7tooltip {
border-radius: 3px 3px 3px 3px;
border-color: #999999;
background-color: #000000;
color: #FFFFFF;
font-family: Arial, Helvetica, sans-serif;
font-size: 11px;
}
```

Twitter share

Twitter share tool consists of a button added to the Social share panel. When the button is clicked the user is redirected to a sharing dialog box that is provided by a social service. The position of the button is fully managed by the Social share tool.

The appearance of the Twitter share button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7twittershare
```

CSS properties of the Twitter share tool

width	Button width.
height	Button height.

background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Twitter share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```
.s7ecatalogsearchviewer .s7twittershare {
  width:28px;
  height:28px;
}
.s7ecatalogsearchviewer .s7twittershare[state='up'] {
background-image:url(images/v2/TwitterShare_dark_up.png);
}
.s7ecatalogsearchviewer .s7twittershare[state='over'] {
background-image:url(images/v2/TwitterShare_dark_over.png);
}
.s7ecatalogsearchviewer .s7twittershare[state='down'] {
background-image:url(images/v2/TwitterShare_dark_down.png);
}
.s7ecatalogsearchviewer .s7twittershare[state='disabled'] {
background-image:url(images/v2/TwitterShare_dark_disabled.png);
}
```

View All Favorites button

The position of the button is fully managed by the Favorites menu.

The appearance of the View All Favorites button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7viewallfavoritebutton
```

CSS properties of the Remove Favorite button

background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .
width	Width of the button.
height	Height of the button.



Note: This button supports both the `state` and `selected` attribute selectors, which can be used to apply different skins to different button states. In particular, `selected='true'` corresponds to the state when a user can add a new Favorite

icon by clicking or tapping. selected='false' corresponds to the normal operation mode when a user can zoom, pan, and swap pages.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a View All favorites button that is 28 x 28 pixels, and displays a different image for each of the four different button states when selected or not selected.

```
.s7ecatalogsearchviewer .s7viewallfavoritebutton {
  width:28px;
  height:28px;
}
.s7ecatalogsearchviewer .s7viewallfavoritebutton[selected='false'][state='up'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7viewallfavoritebutton[selected='false'][state='over'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7viewallfavoritebutton[selected='false'][state='down'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7viewallfavoritebutton[selected='false'][state='disabled'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_disabled.png);
}
.s7ecatalogsearchviewer .s7viewallfavoritebutton[selected='true'][state='up'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7viewallfavoritebutton[selected='true'][state='over'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7viewallfavoritebutton[selected='true'][state='down'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7viewallfavoritebutton[selected='true'][state='disabled'] {
background-image:url(images/v2/ViewAllFavoritesButton_dark_disabled.png);
}
```

Zoom in button

Clicking or tapping this button zooms in on an image in the main view. This button appears in the main control bar. This button is not displayed on mobile phones to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7zoominbutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.
left	Position from the left border of the main control bar, including padding.
bottom	Position from the bottom border of the main control bar, including padding.

CSS property	Description
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom in button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 103 pixels from the right edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7zoominbutton {
  bottom:4px;
  right:103px;
  width:28px;
  height:28px;
}
.s7ecatalogsearchviewer .s7zoominbutton [state='up'] {
  background-image:url(images/v2/ZoomInButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7zoominbutton [state='over'] {
  background-image:url(images/v2/ZoomInButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7zoominbutton [state='down'] {
  background-image:url(images/v2/ZoomInButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7zoominbutton [state='disabled'] {
  background-image:url(images/v2/ZoomInButton_dark_disabled.png);
}
```

Zoom out button

Clicking or tapping this button zooms out on an image in the main view. This button does not display on mobile phones to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7zoomoutbutton
```

CSS property	Description
top	Position from the top border of the main control bar, including padding.
right	Position from the right border of the main control bar, including padding.

CSS property	Description
left	Position from the left border of the main control bar, including padding.
bottom	Position from the bottom border of the main control bar, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom out button that is 28 x 28 pixels, positioned 4 pixels from the bottom and 75 pixels from the right edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7zoomoutbutton {
  bottom:4px;
  right:75px;
  width:28px;
  height:28px;
}
.s7ecatalogsearchviewer .s7zoomoutbutton [state='up'] {
  background-image:url(images/v2/ZoomOutButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7zoomoutbutton [state='over'] {
  background-image:url(images/v2/ZoomOutButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7zoomoutbutton [state='down'] {
  background-image:url(images/v2/ZoomOutButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7zoomoutbutton [state='disabled'] {
  background-image:url(images/v2/ZoomOutButton_dark_disabled.png);
}
```

Zoom reset button

Clicking or tapping this button resets an image in the main view. This button appears in the main control bar on desktop systems and tablets. On mobile phones, this button shows in the bottom center over the image. However, it is not displayed when the image is in a reset state. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7zoomresetbutton
```

CSS property	Description
top	Position from the top border of the main control bar (on desktops and tablets) or viewer (on mobile phones), including padding.
right	Position from the right border of the main control bar (on desktops and tablets) or viewer (on mobile phones), including padding.
left	Position from the left border of the main control bar (on desktops and tablets) or viewer (on mobile phones), including padding.
bottom	Position from the bottom border of the main control bar (on desktops and tablets) or viewer (on mobile phones), including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom reset button that is 28 x 28 pixels, positioned (on desktop) 4 pixels from the bottom and 47 pixels from the right edge of the main control bar, and displays a different image for each of the four different button states.

```
.s7ecatalogsearchviewer .s7zoomresetbutton {
bottom:4px;
right:47px;
width:28px;
height:28px;
}
.s7ecatalogsearchviewer .s7zoomresetbutton [state='up'] {
background-image:url(images/v2/ZoomResetButton_dark_up.png);
}
.s7ecatalogsearchviewer .s7zoomresetbutton [state='over'] {
background-image:url(images/v2/ZoomResetButton_dark_over.png);
}
.s7ecatalogsearchviewer .s7zoomresetbutton [state='down'] {
background-image:url(images/v2/ZoomResetButton_dark_down.png);
}
.s7ecatalogsearchviewer .s7zoomresetbutton [state='disabled'] {
background-image:url(images/v2/ZoomResetButton_dark_disabled.png);
}
```

Support for Adobe Analytics tracking

The eCatalog Search Viewer supports Adobe Analytics tracking out of the box.

Out-of-the-box tracking

The eCatalog Search Viewer supports Adobe Analytics tracking out-of-the-box. To enable tracking, pass the proper company preset name as `config2` parameter.

The viewer also sends a single tracking HTTP request to the configured Image Server with the viewer type and version information.

Custom tracking

To integrate with third-party analytics systems it is necessary to listen to the `trackEvent` viewer callback and process the `eventInfo` argument of the callback function as necessary. The following code is an example of such handler function:

```
var eCatalogSearchViewer = new s7viewers.eCatalogSearchViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Viewers/Pluralist",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  },
  "handlers": {
    "trackEvent": function(objID, compClass, instName, timeStamp, eventInfo) {
      //identify event type
      var eventType = eventInfo.split(",")[0];
      switch (eventType) {
        case "LOAD":
          //custom event processing code
          break;
        //additional cases for other events
      }
    }
  }
});
```

The viewer tracks the following SDK user events:

SDK user event	Sent when...
LOAD	viewer is loaded first.
SWAP	an asset is swapped in the viewer using the <code>setAsset()</code> API.
ZOOM	an image is zoomed.
PAN	an image is panned.
SWATCH	an image is change by clicking or tapping on a swatch.
PAGE	a current frame is changed in the main view.
ITEM	an info panel pop-up is activated.

SDK user event	Sent when...
HREF	a user navigates to a different page because of clicking the image map.

Localization of user interface elements

Certain content that the eCatalog Viewer displays is subject to localization, including zoom buttons, page change buttons, thumbnail button, full screen button, close button, and scroll bar buttons.

Every textual content in the viewer that can be localized is represented by a special Viewer SDK identifier called SYMBOL. Any SYMBOL has a default associated text value for the English locale ("en") supplied with the out-of-the-box viewer, and also may have user-defined values set for as many locales as needed.

When the viewer starts, it checks the current locale to see if there is a user-defined value for each supported SYMBOL in the locale. If there is, it uses the user-defined value; otherwise, it falls back to the out-of-the-box default text.

User-defined localization data can be passed to the viewer as a localization JSON object. Such an object contains the list of supported locales, SYMBOL text values for each locale, and the default locale.

An example of such localization object:

```
{
  "en": {
    "CloseButton.TOOLTIP": "Close",
    "ZoomInButton.TOOLTIP": "Zoom In"
  },
  "fr": {
    "CloseButton.TOOLTIP": "Fermer",
    "ZoomInButton.TOOLTIP": "Agrandir"
  },
  defaultLocale: "en"
}
```

In the example above, the localization object defines two locales ("en" and "fr") and provides localization for two user interface elements in each locale.

The web page code should pass such localization object to the viewer constructor as a value of `localizedTexts` field of the configuration object. An alternative option is to pass the localization object by calling `setLocalizedTexts(localizationInfo)` method.

The following SYMBOLs are supported (assuming `containerId` is theID of the viewer container):

SYMBOL	Tool tip for...
CloseButton.TOOLTIP	Close button.
ZoomInButton.TOOLTIP	Zoom in button.
ZoomOutButton.TOOLTIP	Zoom out button.
ZoomResetButton.TOOLTIP	Zoom reset button.
FullScreenButton.TOOLTIP_SELECTED	Full screen button in normal state.

SYMBOL	Tool tip for...
FullScreenButton.TOOLTIP_UNSELECTED	Full screen button in full screen state.
ScrollUpButton.TOOLTIP	Scroll up button.
ScrollDownButton.TOOLTIP	Scroll down button.
<containerId>_rightButton.PanRightButton.TOOLTIP	Large next page button.
<containerId>_leftButton.PanLeftButton.TOOLTIP	Large previous page button.
<containerId>_lastPageButton.PanRightButton.TOOLTIP	Last page button.
<containerId>_secondaryLastPageButton.PanRightButton.TOOLTIP	Last page button.
<containerId>_firstPageButton.PanLeftButton.TOOLTIP	First page button.
<containerId>_secondaryFirstPageButton.PanLeftButton.TOOLTIP	First page button.
<containerId>_toolBarRightButton.PanRightButton.TOOLTIP	Next page button.
<containerId>_toolBarLeftButton.PanLeftButton.TOOLTIP	Previous page button.
ThumbnailPageButton.TOOLTIP_SELECTED	Thumbnails button in thumbnails mode.
ThumbnailPageButton.TOOLTIP_UNSELECTED	Thumbnails button in normal mode.
CloseButton.TOOLTIP	Close button.
InfoPanelPopup.TOOLTIP_CLOSE	Info Panel close button.
SocialShare.TOOLTIP	Social share tool.
EmailShare.TOOLTIP	Email share button.
EmailShare.HEADER	Email dialog header.
EmailShare.TOOLTIP_HEADER_CLOSE	Email dialog box upper-right close button.
EmailShare.INVALID_ADDRESSSS	Error message displayed in case email address is malformed.

SYMBOL	Tool tip for...
EmailShare.TO	Label for the "To" input field.
EmailShare.TOOLTIP_ADD	Add Another Email Address button.
EmailShare.ADD	Add Another Email Address button.
EmailShare.FROM	From input field.
EmailShare.MESSAGE	Message input field.
EmailShare.TOOLTIP_REMOVE	Remove Email Address button.
EmailShare.CANCEL	Caption for the Cancel button.
EmailShare.TOOLTIP_CANCEL	Cancel button.
EmbedShare.ACTION	Caption for the Select All button.
EmbedShare.TOOLTIP_ACTION	Select All button.
EmailShare.CLOSE	Caption for the close button displayed in the bottom of dialog after form submission.
EmailShare.TOOLTIP_CLOSE	Close button that is displayed in the bottom of dialog after form submission.
EmailShare.ACTION	Caption for the form submission button.
EmailShare.TOOLTIP_ACTION	Form submission button.
EmailShare.SEND_SUCCESS	Confirmation message displayed when email was sent successfully.
EmailShare.SEND_FAILURE	Error message that is displayed when email was not sent successfully.
EmbedShare.TOOLTIP	Embed share button.
EmbedShare.HEADER	Embed dialog box header.

SYMBOL	Tool tip for...
EmbedShare.TOOLTIP_HEADER_CLOSE	Embed dialog box upper-right close button.
EmbedShare.DESCRPTION	Description of the embed code text.
EmbedShare.EMBED_SIZE	Label for the embed size combo box.
EmbedShare.CANCEL	Caption for the Cancel button.
EmbedShare.TOOLTIP_CANCEL	Cancel button.
EmbedShare.CUSTOM_SIZE	Text for the last "custom size" entry in the embed size combo box.
LinkShare.TOOLTIP	Link share button.
LinkShare.HEADER	Link dialog box header.
LinkShare.TOOLTIP_HEADER_CLOSE	Link dialog box upper-right close button.
LinkShare.DESCRPTION	Description of the share link.
LinkShare.CANCEL	Caption for the Cancel button.
LinkShare.TOOLTIP_CANCEL	Cancel button.
LinkShare.ACTION	Caption for the Select All button.
LinkShare.TOOLTIP_ACTION	Select All button.
FacebookShare.TOOLTIP	Facebook share button.
TwitterShare.TOOLTIP	Twitter share button.
Print.TOOLTIP	Print button.
Print.HEADER	Print dialog header.
Print.TOOLTIP_HEADER_CLOSE	Print dialog box top right close button.
Print.PRINT_RANGE	Label for the "Select Print Pages" section.

SYMBOL	Tool tip for...
Print.PRINT_RANGE_CURRENT	Caption for the "Current pages" radio button.
Print.PRINT_RANGE_FROM	Caption for the "Spread range from" radio button.
Print.PRINT_RANGE_TO	Caption for the "to" numeric picker.
Print.PRINT_RANGE_ALL	Caption for the "All pages" radio button.
Print.PAGE_HANDLING	Label for the "Page Handling" section.
Print.PAGE_HANDLING_ONE	Caption for the "1 page per sheet" radio button.
Print.PAGE_HANDLING_TWO	Caption for the "2 pages per sheet" radio button.
Print.CANCEL	Caption for the Cancel button.
Print.TOOLTIP_CANCEL	Cancel button.
Print.ACTION	Caption for the Send to print button
Print.TOOLTIP_ACTION	Send to print button.
FavoritesMenu.TOOLTIP	Favorites menu button.
AddFavoriteButton.TOOLTIP_SELECTED	"Add favorite" button in edit Favorites mode.
AddFavoriteButton.TOOLTIP_UNSELECTED	"Add favorite" button in normal mode.
RemoveFavoriteButton.TOOLTIP_SELECTED	"Remove favorite" button in edit Favorites mode.
RemoveFavoriteButton.TOOLTIP_UNSELECTED	"Remove favorite" button in normal mode.
ViewAllFavoriteButton.TOOLTIP_SELECTED	"View all favorites" button when Favorites view is active.

SYMBOL	Tool tip for...
<code>ViewAllFavoriteButton.TOOLTIP_UNSELECTED</code>	"View all favorites" button when Favorites view is inactive.
<code>FavoritesEffect.TOOLTIP</code>	Single favorite icon.
<code>MediaSet.LABEL_XX[_YY]</code>	<p>Page label that is generated by the viewer at load time.</p> <p>The name of that symbol is a template, where XX is a zero-based spread index in landscape orientation, and optional YY is a zero-based page index inside the spread targeted by XX.</p> <p>Applies only for the initially loaded asset; ignored if an asset is changed using the <code>setAsset()</code> API call.</p>
<code>MediaSet.LABEL_DELIM</code>	Character used as a page labels delimiter in case labels are defined for left and right pages within a spread.
<code>ScrollLeftRightButton.TOOLTIP_SELECTED</code>	Main control bar scroll left button.
<code>ScrollLeftRightButton.TOOLTIP_UNSELECTED</code>	Main control bar scroll right button.
<code>SearchPanel.PLACEHOLDER</code>	Localized prompt shown inside search input box before user starts entering the search text.
<code>SearchPanel.INFO_PROMPT</code>	Localized message shown when the search panel is opened for the first time, suggesting user perform the search.
<code>SearchPanel.INFO_NO_RESULTS</code>	<p>Localized message shown when the search did not return any results.</p> <p>This symbol supports the following runtime replacement token: <code>\$SEARCH_TEXT\$</code>. The component replaces it with the search text entered by the user.</p>

SYMBOL	Tool tip for...
<code>SearchPanel.INFO_RESULTS</code>	<p>Localized message shown when the search is successfully completed and returns at least one result.</p> <p>This symbol supports the following runtime replacement tokens:</p> <ul style="list-style-type: none"> • <code>\$SEARCH_TEXT\$</code> – The search text entered by the user. • <code>\$HIT_COUNT\$</code> – The total number of search hits found. • <code>\$PAGE_COUNT\$</code> – The number of catalog pages that contain at least one search hit.
<code>SearchPanel.THUMBNAIL_LABEL</code>	<p>Localized label for results thumbnail of search panel.</p> <p>This symbol supports the following runtime replacement tokens:</p> <ul style="list-style-type: none"> • <code>\$PAGE\$</code> – Page number. • <code>\$PAGE_HIT_COUNT\$</code> – The number of search results found on the page.

Image map support

The eCatalog Search Viewer supports the rendering of image map icons above the main view.

The appearance of map icons is controlled through CSS as described in [Image map effect](#).

Image maps perform one of the following three actions: redirect to an external web page, Info panel pop-up activation, and internal hyperlinks.

Redirect to an external web page

The `href` attribute of the image map has a URL to the external resource, either specified explicitly, or wrapped into one of the supported JavaScript template functions: `loadProduct()`, `loadProductCW()`, and `loadProductPW()`.

The following is an example of a simple URL redirect:

```
href=http://www.adobe.com
```

In this example, the same URL is wrapped with the `loadProduct()` function:

```
href=javascript:loadProduct("http://www.adobe.com");void(0);
```

Be aware that when you add the JavaScript code into the `HREF` attribute of your image map, the code is run on the client's computer. Therefore, make sure that the JavaScript code is secure.

Info Panel Popup activation

To work with Info panels, an image map has the `ROLLOVER_KEY` attribute set. Also, set the `href` attribute at the same time, otherwise the external URL processing interferes with the Info panel pop-up activation.

Finally, be sure that the viewer configuration includes the appropriate values for `InfoPanelPopup.template` and, optionally, `InfoPanelPopup.infoServerUrl` parameters.



Note: Be aware that when you configure Info Panel Popup, the HTML code and JavaScript code passed to the Info Panel runs on the client's computer. Therefore, make sure that such HTML code and JavaScript code are secure.

Internal hyperlinks

Clicking on an image map performs an internal page swap inside the viewer. To use that feature, an `href` attribute in the image map has the following special format:

```
href=target:idx
```

where `idx` is a zero-based index of the catalog spread.

The following is an example of an `href` attribute for an image map that points to the 3D spread in the eCatalog:

```
href=target:2
```

Managing page labels

There are two places in the viewer user interface where page labels are shown: thumbnails mode and the table of contents drop-down.

There are three types of labels that can be defined:

- Labels defined by the author using SYMBOL localization mechanism.
- Labels defined by the author on the backend, inside Scene7 Publishing System.
- Labels automatically generated by the viewer.

SYMBOL-based labels are defined using `MediaSet.LABEL_XX[_YY]` and `MediaSet.LABEL_DELIM` SYMBOLs as described in [Localization of user interface elements](#). You can define such labels either for the entire ecatalog spread, in which case you should use short SYMBOL syntax (`MediaSet.LABEL_XX`). Or, specify it for each page individually using full SYMBOL syntax (`MediaSet.LABEL_XX_YY`).

When you define labels for both pages in the ecatalog spread, the viewer concatenates these labels into one string using `MediaSet.LABEL_DELIM` SYMBOL. SYMBOL-based labels take precedence over labels that are defined on the backend or automatically generated labels by the viewer.

Labels defined in Scene7 Publishing System are stored in the `UserData` record of individual page images. Same as with SYMBOL-based labels. That is, if both pages in the ecatalog spread have labels defined, they are concatenated using `MediaSet.LABEL_DELIM` SYMBOL in landscape mode. Scene7 Publishing System labels take precedence over automatically generated labels, but are overridden by SYMBOL-based labels.

Automatically generated labels are sequential numbers assigned to all pages in the ecatalog. Automatically generated labels are ignored for the given spread if it has SYMBOL-based labels defined or Scene7 Publishing System labels defined.

In the table of contents, it is possible to disable the display of automatically generated labels using `showdefault` parameter.

Print feature

The viewer lets you output the catalog content to a printer.

The print feature is triggered by a dedicated button in the toolbar. Clicking the button lets the user choose a print range and the number of pages per sheet.

The quality of the print can be adjusted using the `printquality` configuration parameter. Note that setting `printquality` to values significantly higher than the default is not recommended. The reason is because it leads to very high memory consumption by the web browser on the client's system. Also, make sure that the maximum image response size set for your SPS company is larger than the configured `printquality` value.



Note: The Print feature is only available on desktop systems, except Internet Explorer 9.

Full screen support

The viewer supports full screen operation mode.

On modern desktop browsers, except Internet Explorer 10 and older, and on some touch devices, the viewer uses "native" full screen mode. This mode means that the entire device screen is occupied by the viewer content.

On iOS devices and on older Internet Explorer browsers, the viewer uses "simulated" full screen mode instead. In this mode, the viewer simply resizes to take the full area of the web browser window. Also, the web browser Chrome and other windows are still visible on the screen.

An end user enters and leaves full screen mode by pressing the Full Screen button in the viewer user interface. When "native" full screen mode is used on desktop, it is also possible to exit it by pressing **Esc**.

Download

It is possible to download the electronic catalog as a PDF file using the "Download" button in the control bar.

The "Download" button is automatically available in the viewer user interface when the following occurs:

- An actual PDF file is present in the customer's company.
- The name matches the name of the e-catalog asset that is passed to the viewer and includes a `.pdf` extension.
- The PDF file is published in SPS (Scene7 Publishing System).

Favorites feature

The viewer supports text searching over the catalog contents. For the search feature to work the catalog "Extract Search Words" feature must be turned on when the source PDF file is uploaded to Scene7 Publishing System.

The Search feature is triggered by activating a Search button in the main tool bar. This action brings up a search results panel with a text input field. A user can run a search by using keywords that separated with spaces or by using phrases the are surrounded by double quotes.

Search results are displayed as thumbnails of pages where search text was found. Additionally, the viewer shows a total number of pages and hits found during the search. The user can click or tap on a search result thumbnail to navigate to the corresponding page of the catalog.

Finally, the viewer highlights search hits in the main view using semi-transparent yellow regions.

Viewer SDK namespace

The viewer is built of many Viewer SDK components. In most cases, the web page does not need to interact with SDK components API directly; all common needs are covered in the viewer API itself.

However, some advanced use cases require that the web page obtain a reference to an inner SDK component using the `getComponent()` viewer API and then use all the flexibility of the APIs of SDK itself.

The namespace that is used to load and initialize SDK components by the viewer depends on the environment in which the viewer is operating. If the viewer is running in AEM (Adobe Experience Manager), the viewer loads SDK components into `s7viewers.s7sdk` namespace. And the viewer served from Scene7 Publishing System loads the SDK into `s7classic.s7sdk`.

In either case, the namespace used by the SDK inside the viewer has either `s7viewers` or `s7classic` as the prefix. And, it is different from the plain `s7sdk` namespace used in the SDK User Guide or SDK API documentation.

For that reason it is important to use a fully qualified SDK namespace when you write custom application code that communicates with internal viewer components.

For example, if you plan to listen to `StatusEvent.NOTF_VIEW_READY` event and the viewer is served from Scene7 Publishing System, the fully qualified event type is `s7classic.s7sdk.event.StatusEvent.NOTF_VIEW_READY`, and the event listener code looks similar to the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var pageView = <instance>.getComponent("pageView");
    pageView.addEventListener(s7classic.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});
```

Flyout

Flyout Viewer is an image viewer. It displays a static image with the zoomed version shown in the flyout view that a user activates. This viewer works with image sets and navigation is done by using swatches. It is designed to work on desktops and mobile devices.



Note: Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.

Viewer type is 504.

See [System requirements](#).

Demo URL

<https://s7d9.scene7.com/s7viewers/html5/FlyoutViewer.html?asset=Scene7SharedAssets/ImageSet-Views-Sample>

Using Flyout Viewer

Flyout Viewer represents a main JavaScript file and a set of helper files (a single JavaScript include with all Viewer SDK components used by this particular viewer, assets, CSS) downloaded by the viewer in runtime

Flyout Viewer is only intended for embedded use, which means that it is integrated into the web page using documented API. No production-ready web page is available for the Flyout Viewer.

Configuration and skinning are similar to that of the other viewers. You can use custom CSS to apply skinning.

See [Command reference common to all viewers – Configuration attributes](#) and [Command reference common to all Viewers – URL](#)

Interacting with Flyout Viewer

Flyout Viewer supports single-touch and multi-touch gestures that are common in other mobile applications.

Gesture	Description
Single tap	Activates the flyout view or changes between the primary and secondary zoom level.
Horizontal swipe or flick	Scrolls through the list of swatches in the swatch bar.
Vertical swipe	If the gesture is done within the swatches area, it performs a native page scroll.

The viewer also supports both touch input and mouse input on Windows devices with touch screen and mouse. This support, however, is limited to Chrome, Internet Explorer 11, and Edge web browsers only.

This viewer is fully keyboard accessible.

See [Keyboard accessibility and navigation](#).

Embedding Flyout Viewer

Different web pages have different needs for viewer behavior. The web page may have static page layout, or use responsive design that displays differently on different devices, or for different browser window sizes. To accommodate these needs, the viewer supports two primary operation modes: fixed size embedding and responsive design embedding.

Fixed size embedding mode is used when the viewer does not change its size after its initial load. This choice is best for web pages that have a static page layout.

Responsive design embedding mode assumes that the viewer may need to resize during runtime in response to the size change of its container `DIV`. The most common use case is adding a viewer to a web page that uses a flexible page layout.

When using responsive design embedding mode with the Flyout Viewer, make sure that you specify explicit breakpoints for the main view image using the `imagereload` parameter. Ideally, match your breakpoints with the viewer width breakpoints as dictated by the web page CSS.

In responsive design embedding mode the viewer behaves differently depending on the way a web page sizes its container `DIV`. If the web page sets only the width of the container `DIV`, leaving its height unrestricted, then the viewer automatically chooses its height according to the aspect ratio of the asset that is used. This means that the asset fits perfectly into the view without any padding on the sides. This particular use case is the most common for web pages that use responsive design layout frameworks like Bootstrap, Foundation, and so forth.

Otherwise, if the web page sets both the width and the height for the viewer's container `DIV`, then the viewer fills only that area and follows the size provided by the web page layout. A good use case example is embedding the viewer into a modal overlay, where the overlay is sized according to web browser window size.

Fixed size embedding

You add the viewer to a web page by doing the following:

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Setting the viewer size.
4. Creating and initializing the viewer.

1. Adding the viewer JavaScript file to your web page.

Creating a viewer requires that you add a script tag in the HTML head. Before you can use the viewer API, be sure that you include `FlyoutViewer.js`. `FlyoutViewer.js` is located in the following `html5/js/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/js/FlyoutViewer.js
```

You can use a relative path if the viewer is deployed on one of the Adobe Scene7 servers and it is served from the same domain. Otherwise, you specify a full path to one of Adobe Scene7 servers that have the IS-Viewers installed.

A relative path looks like the following:

```
<script language="javascript" type="text/javascript"
src="/s7viewers/html5/js/FlyoutViewer.js"></script>
```



Note: You should only reference the main viewer JavaScript include file on your page. You should not reference any additional JavaScript files in the web page code which might be downloaded by the viewer's logic in runtime. In particular, do not directly reference HTML5 SDK `Utils.js` library loaded by the viewer from `/s7viewers` context path (so-called consolidated SDK include). The reason is that the location of `Utils.js` or similar runtime viewer libraries is fully managed by the viewer's logic and the location changes between viewer releases. Adobe does not keep older versions of secondary viewer includes on the server.

As a result, putting a direct reference to any secondary JavaScript include used by the viewer on the page breaks the viewer functionality in the future when a new product version is deployed.

2. Defining the container DIV.

Add an empty DIV element to the page where you want the viewer to appear. The DIV element must have its ID defined because this ID is later passed to the viewer API.

The placeholder DIV is a positioned element, meaning that the `position` CSS property is set to `relative` or `absolute`.

It is the responsibility of the web page to specify the proper `z-index` for the placeholder DIV element. Doing so ensures that the viewer's flyout portion appears on top of the other web page elements.

The following is an example of a defined placeholder DIV element:

```
<div id="s7viewer" style="position:relative;z-index:1"></div>
```

3. Setting the viewer size.

This viewer displays thumbnails when working with multi-item sets. On desktop systems, thumbnails are placed below the main view. At the same time, the viewer allows the swapping of the main asset during runtime using `setAsset()` API. As a developer, you have control over how the viewer manages the thumbnails area in the bottom area when the new asset has only one item. It is possible to keep the outer viewer size intact and let the main view increase its height and occupy the thumbnails area. Or, you can keep the main view size static and collapse the outer viewer area, thereby letting web page content to move up, and then use the free page real estate left from the thumbnails.

To keep the outer viewer bounds intact define the size for `.s7flyoutviewer` top-level CSS class in absolute units. Sizing in CSS can be put right on the HTML page, or in a custom viewer CSS file, which is later assigned to a viewer preset record in Scene7 Publishing System, or passed explicitly using style command.

See [Customizing Flyout Viewer](#) for more information about styling the viewer with CSS.

The following is an example of defining the static outer viewer size in an HTML page:

```
#s7viewer.s7flyoutviewer {
  width: 640px;
  height: 480px;
}
```

You can see the behavior with a fixed outer viewer area on the following sample page. Notice that when you switch between sets, the outer viewer size does not change:

https://marketing.adobe.com/resources/help/en_US/s7/viewers_ref/samples/FlyoutViewer-fixed-outer-area.html

To make the main view dimensions static, define the viewer size in absolute units for the inner `Container` SDK component using `.s7flyoutviewer .s7container` CSS selector. In addition, you should override the fixed size defined for the `.s7flyoutviewer` top-level CSS class in the default viewer CSS, by setting it to `auto`.

The following is an example of defining the viewer size for the inner `Container` SDK component so that the main view area does not change its size when switching the asset:

```
#s7viewer.s7flyoutviewer {
  width: auto;
  height: auto;
}
#s7viewer.s7flyoutviewer .s7container {
  width: 640px;
  height: 480px;
}
```

The following sample page shows viewer behavior with a fixed main view size. Notice that when you switch between sets, the main view remains static and the web page content moves vertically:

https://marketing.adobe.com/resources/help/en_US/s7/viewers_ref/samples/FlyoutViewer-fixed-main-view.html

Also, note that the default viewer CSS provides a fixed size for its outer area out-of-the-box.

4. Creating and initializing the viewer.

When you have completed the steps above, you create an instance of `s7viewers.FlyoutViewer` class, pass all configuration information to its constructor and call `init()` method on a viewer instance. Configuration information is passed to the constructor as a JSON object. At minimum, this object should have the `containerId` field which holds the name of viewer container ID and nested `params` JSON object with configuration parameters that the viewer supports. In this case, the `params` object must have at least the Image Serving URL passed as `serverUrl` property, and the initial asset as `asset` parameter. JSON-based initialization API lets you create and start the viewer with single line of code.

It is important to have the viewer container added to the DOM so that the viewer code can find the container element by its ID. Some browsers delay building DOM until the end of the web page. For maximum compatibility, call the `init()` method just before the closing `BODY` tag, or on the `body onload()` event.

At the same time, the container element should not necessarily be part of the web page layout just yet. For example, it may be hidden using `display:none` style assigned to it. In this case, the viewer delays its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load automatically resumes.

The following is an example of creating a viewer instance, passing the minimum necessary configuration options to the constructor and calling the `init()` method. The example assumes `flyoutViewer` is the viewer instance; `s7viewer` is the name of placeholder DIV; `http://s7d1.scene7.com/is/image/` is the Image Serving URL; and `Scene7SharedAssets/ImageSet-Views-Sample` is the asset:

```
<script type="text/javascript">
var flyoutViewer = new s7viewers.FlyoutViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
    "serverurl": "http://s7d1.scene7.com/is/image/"
  }
}).init();
</script>
```

The following code is a complete example of a trivial web page that embeds the Flyout Viewer with a fixed size:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7d1.scene7.com/s7viewers/html5/js/FlyoutViewer.js"></script>
<style type="text/css">
#s7viewer.s7flyoutviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative;z-index:1;"></div>
<script type="text/javascript">
var flyoutViewer = new s7viewers.FlyoutViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
    "serverurl": "http://s7d1.scene7.com/is/image/"
  }
}).init();
</script>
</body>
</html>
```

Responsive design embedding with unrestricted height

With responsive design embedding, the web page normally has some kind of flexible layout in place that dictates the runtime size of the viewer's container DIV. For the following example, assume that the web page allows the viewer's container DIV to take 40% of the web browser window size, leaving its height unrestricted. The web page HTML code would look like the following:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
.holder {
  width: 40%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>
```

Adding the viewer to such a page is similar to the steps for fixed size embedding. The only difference is that you need to override the fixed sizing from the default viewer CSS with the size set in relative units.

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Setting the viewer size.
4. Creating and initializing the viewer.

All the steps above are the same as with the fixed size embedding with the following three exceptions:

- add the container DIV to the existing "holder" DIV;
- added `imagereload` parameter with explicit breakpoints;
- instead of setting a fixed viewer size using absolute units use CSS that sets the viewer width and height to 100% as in the following:

```
#s7viewer.s7flyoutviewer {
  width: 100%;
  height: 100%;
}
```

The following code is a complete example. Notice how the viewer size changes when the browser is resized, and how the viewer aspect ratio matches the asset.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/FlyoutViewer.js"></script>
<style type="text/css">
.holder {
  width: 40%;
}
#s7viewer.s7flyoutviewer {
  width: 100%;
  height: 100%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative;z-index:1"></div>
</div>
<script type="text/javascript">
var flyoutViewer = new s7viewers.FlyoutViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Scene7SharedAssets/ImageSet-Views-Sample",
    "serverurl":"http://s7dl.scene7.com/is/image/",
    "imagereload":"1,breakpoint,200;400;800;1600"
  }
}).init();
</script>
</body>
</html>
```

The following examples page illustrates more real-life uses of responsive design embedding with unrestricted height:

https://marketing.adobe.com/resources/help/en_US/s7/vlist/vlist.html

Flexible size embedding with width and height defined

In case of flexible-size embedding with width and height defined, the web page styling is different. It provides both sizes to the "holder" DIV and centers it in the browser window. Also, the web page sets the size of the HTML and BODY element to 100 percent.

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>
```

The rest of the embedding steps are identical to the steps used for responsive design embedding with unrestricted height. The resulting example is the following:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/FlyoutViewer.js"></script>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
#s7viewer.s7flyoutviewer {
  width: 100%;
  height: 100%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative;z-index:1"></div>
</div>
<script type="text/javascript">
var flyoutViewer = new s7viewers.FlyoutViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Scene7SharedAssets/ImageSet-Views-Sample",
    "serverurl":"http://s7dl.scene7.com/is/image/",
    "imagereload":"1,breakpoint,200;400;800;1600"
  }
}).init();
</script>
```



```
</body>
</html>
```

Embedding using Setter-based API

Instead of using JSON-based initialization, it is possible to use setter-based API and no-args constructor. Using this API constructor does not take any parameters and configuration parameters are specified using `setContainerId()`, `setParam()`, and `setAsset()` API methods, with separate JavaScript calls.

The following example illustrates using fixed size embedding with setter-based API:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/FlyoutViewer.js"></script>
<style type="text/css">
#s7viewer.s7flyoutviewer {
  width: 640px;
  height: 480px;
}
</style>
</head><body>
<div id="s7viewer" style="position:relative;z-index:1;"></div>
<script type="text/javascript">
var flyoutViewer = new s7viewers.FlyoutViewer();
flyoutViewer.setContainerId("s7viewer");
flyoutViewer.setParam("serverurl", "http://s7dl.scene7.com/is/image/");
flyoutViewer.setAsset("Scene7SharedAssets/ImageSet-Views-Sample");
flyoutViewer.init();
</script>
</body>
</html>
```

Command reference – Configuration attributes

Configuration attributes documentation for Flyout Viewer

You can set any configuration command in URL. Or, you can use `setParam()`, `setParams()`, or both API methods. You can also specify any configuration attribute in the server-side configuration record.

Some configuration commands are prefixed with the class name or instance name of corresponding Viewer SDK component. An instance name of the component is dynamic and depends on the ID of the viewer container DOM element passed to `setContainerId()` API method. Documentation includes an optional prefix for such commands. For example, the `zoomfactor` command is documented as follows:

```
[FlyoutZoomView.|<containerId>_flyout].zoomfactor
```

The command is used as follows:

- `zoomfactor` (short syntax)
- `FlyoutZoomView.zoomfactor` (qualified with a component class name)
- `cont_flyout.zoomfactor` (qualified with the component ID, assuming that `cont` is the ID of the container element)

See also [Command reference common to all viewers – Configuration attributes](#)

FlyoutZoomView.flyouttransition

```
[FlyoutZoomView.|<containerId>_flyout.flyouttransition=[none|slide|fade][,showtime[,showdelay[,hidetime[,hidedelay]]]]
```

<i>none</i> / <i>slide</i> / <i>fade</i>	Specifies the type the effect applied when the flyout view is shown or hidden. With <i>none</i> , the flyout image appears instantly when activated and ready; <i>slide</i> makes the slide animation play in left-to-right direction; <i>fade</i> applies an alpha transition to the flyout image.
<i>showtime</i>	Number of seconds that the show animation takes to complete.
<i>showdelay</i>	The delay in seconds between user action which initiates the show animation and the beginning of show animation itself.
<i>hidetime</i>	Number of seconds that the hide animation takes to complete.
<i>hidedelay</i>	The delay in seconds between user action which initiates the hide animation and the beginning of hide animation itself.

Properties

Optional.

Default

`fade,1,0,1,0`

Example

`flyouttransition=slide,1,1,2,1`

FlyoutZoomView.fmt

[FlyoutZoomView. | <containerId>_flyout.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha

jpg jpeg png png-alpha gif gif-alpha	Specifies image format to be used by the component for loading images from Image Server. If the specified format ends with "-alpha", the component renders images as transparent content. For all other image formats the component treats images as opaque. Note that the component has a white background by default. Therefore, to make it completely transparent set the background-color CSS property to transparent.
--	--

Properties

Optional.

Default

`jpeg`

Example

`fmt=png-alpha`

FlyoutZoomView.frametransition

```
[FlyoutZoomView. | <containerId>_flyout. ]frametransition=none|fade[ ,duration]
```

none fade	Specifies the type of the effect applied to main view on asset change. The <code>none</code> stands for no transition, main view change happens instantly. The <code>fade</code> activates cross-fade transition where the old image fades out and the new image fades in
<i>duration</i>	Number of seconds that the animation takes to complete.

Properties

Optional.

Default

None.

Example

```
frametransition=fade,1
```

FlyoutZoomView.highlightmode

```
[FlyoutZoomView. | <containerId>_flyout. ]highlightmode=highlight|cursor[ ,showtime[ ,onimage|free]]
```

highlight cursor	Specifies the type of navigation frame to use. When set to <code>cursor</code> , the component uses a fixed-size reference cursor. It is possible to have different cursor art for desktop systems and touch devices, this is controlled with <code>.s7cursor</code> CSS class and <code>input=mouse touch</code> attribute selector. On desktop systems, an anchor point is set in the middle of the cursor area, while on touch devices anchor is located in the bottom center of the cursor. When set to <code>highlight</code> , the component uses a variable-size navigation frame; the size and shape of the frame depends on the zoom factor and the size of the flyout view.
<i>showtime</i>	<p>Sets the time (in seconds) it takes the highlight or cursor to fade in after it is activated by the user. Fade in is applied only on touch devices; on desktop systems it is ignored by the component.</p> <p>Fade in applies to the following UI elements: highlight frame, fixed cursor, overlay (in case <code>overlay</code> parameter is set to 1). Flyout view animation begins only after highlight/cursor fade in animation completes. There is no fade out animation. When the user deactivates the flyout, corresponding UI elements (cursor, highlight and overlay) hide instantly.</p>
onimage free	<p>Controls navigation frame positioning.</p> <p>If set to <code>onimage</code> the navigation frame can only be positioned inside the actual image area within the main view.</p>

	If set to <code>free</code> a user can move the navigation frame anywhere in the logical main view area, even outside image content.
--	--

Properties

Optional.

Default

`highlight,0.1,onimage`

Example

`highlightmode=cursor,1,free`

FlyoutZoomView.imagereload

`[FlyoutZoomView.<containerId>_flyout.]imagereload=0|1[,breakpoint,width[:width]]`

<code>0 1</code>	Configures how the component fetches new images for the main and flyout view during resize. When set to 0, the component does not load new images during resize; image resolution in the flyout view does not change. Setting to 1 lets you specify one or more width breakpoints for the image loaded into the main view.
<code>breakpoint,width[:width]</code>	Width breakpoints for the image that is loaded into the main view. The component always uses the best fit size for the initial load. After resize, it ensures that the image in the main view is always downloaded with the width equal to closest larger breakpoint, and downscaled on the client.

Properties

Optional.

Default

0

Example

`imagereload=1,breakpoint,200;400;800;1600`

FlyoutZoomView.iscommand

`[FlyoutZoomView.<containerId>_flyout.]iscommand=isCommand`

<code>isCommand</code>	The Image Serving command string that is applied to the FlyoutZoomView main image and the zoomed in view. If it is specified in the URL, be sure that you HTTP-encode all occurrences of <code>&</code> and <code>=</code> as <code>%26</code> and <code>%3D</code> , respectively.
------------------------	---



Note: Image sizing manipulation commands are not supported.

Properties

Optional.

Default

None.

Example

When specified in the viewer URL:

```
iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000
```

When specified in the configuration data:

```
iscommand=op_sharpen=1&op_colorize=0xff0000
```

FlyoutZoomView.overlay

```
[FlyoutZoomView. | <containerId>_flyout. ]overlay=0|1
```

0 1	Controls the main view highlight appearance when the flyout is active. When set to 0, the area currently visible in the flyout window is highlighted using styles provided by either <code>.s7highlight</code> or <code>.s7cursor</code> CSS class names (depending on the value of <code>highlightmode</code> modifier). When set to 1 component enters "inverse" mode where the currently viewed area is either fully transparent (in case <code>highlightmode</code> is set to <code>highlight</code>) or styled with <code>.s7cursor</code> CSS class name (in case <code>highlightmode</code> is set to <code>cursor</code>), but surrounding area is filled using styles provided by <code>.s7overlay</code> CSS class name.
-----	--

Properties

Optional.

Default

0


Example

```
overlay=1
```

FlyoutZoomView.preloadtiles

```
[FlyoutZoomView. | <containerId>_flyout. ]preloadtiles=0|1
```

0 1	Set to 1 to enable preloading of the zoomed image, or set to 0 to load the zoom image incrementally, as needed.
-----	---

 **Note:** If you enable this option it can result in substantially higher bandwidth usage. The zoomed image is loaded in its entirety, even if the user does not initiate a zoom action.

Properties

Optional.

Default

0

Example

preloadtiles=1

FlyoutZoomView.tip

[FlyoutZoomView. | <containerId>_flyout.]tip=duration[,count][,fade]

<i>duration</i>	Specifies the number of seconds the tip text is displayed before it hides. When set to -1, the message is always displayed, even if the user activates the flyout.
<i>count</i>	Specifies the number of times the text is displayed when viewing new images in the set. A value of -1 means that the text is always displayed when viewing any image in the set.
<i>fade</i>	Specifies the duration of a fade animation that occurs when the text appears or disappears. A value of 0 means no fade transition.

Properties

Optional.

Default

3,1,0.3

Example

tip=1,-1,0

FlyoutZoomView.zoomfactor

[FlyoutZoomView. | <containerId>_flyout.]zoomfactor=primaryFactor[, [secondaryFactor][,upscale]]

<i>primaryFactor</i>	Specifies the image magnification for the flyout view, relative to the main view. Must be an integer or floating point value 1.0 or larger.
----------------------	---

<i>secondaryFactor</i>	An optional secondary factor can be specified which is accessible by clicking or tapping on the main view when the highlight is active. Clicking or tapping a second time reverts to the primary zoom factor. A value of -1 disables the secondary zoom factor.
<i>upscale</i>	<p>Specifies how the component handles small images.</p> <p>If set to 1 the component upscales the main image so that it fits within the main view. Also, it upscales the zoom image so that it completely fills the configured flyout window area.</p> <p>If set to 0 small images are displayed at their original resolution and display centered in the main view area and inside the flyout window. You can configure extra white space that appears around the image with a background or similar CSS property of the <code>s7flyoutzoomview</code> and <code>s7flyoutzoom</code> CSS classes in the main view and flyout window, respectively.</p>

Properties

Optional.

Default

3,-1,1

Example

zoomfactor=2,3,0

Swatches.align

[Swatches.<containerId>_swatches.]align=left|center|right,top|center|bottom

Specifies internal alignment (anchoring) of the swatches container within the component area. In Swatches, the internal thumbnail container is sized so that only a whole number of swatches is shown. As a result, there is some padding between the internal container and the external component bounds. This command specifies how the internal swatches container is positioned inside the component.

left center right	Sets horizontal swatches alignment.
top center bottom	Sets vertical swatches alignment.

Properties

Optional.

Default

center,center

Example

align=left,top

Swatches.buttonsnapmode

```
[Swatches.|<containerId>_swatches.]buttonsnapmode=snapin|snapout|overlay
```

<i>snapin</i>	Causes the buttons to align next to the swatches.
<i>snapout</i>	Causes the buttons to align next to the component border.
<i>overlay</i>	Causes the buttons to render on top of the swatches.

Properties

Optional.

Default

snapout

Example

```
buttonsnapmode=overlay
```

Swatches.direction

```
[Swatches.|<containerId>_swatches.]direction=auto|left|right
```

<i>auto left right</i>	<p>Specifies the way swatches fill in the view.</p> <p><i>left</i> sets left-to-right fill order; <i>right</i> reverses the order so that the view is filled in right-to-left, top-to-bottom direction. When <i>auto</i> is set, the component applies right mode when locale is set to "ja", and uses left otherwise.</p>
------------------------	--

Properties

Optional.

Default

auto

Example

```
direction=right
```

Swatches.enabledragging

```
[Swatches.|<containerId>_swatches.]enabledragging=0|1[,overdragvalue]
```


0 1	Enables or disables the ability for a user to scroll the swatches with a mouse or by using touch gestures
<i>overdragvalue</i>	Functions within the 0–1 range. It is a % value for movement in the wrong direction of the actual speed. If it is set to 1, it moves with the mouse. If it is set to 0, it does not let you move in the wrong direction at all.

Properties

Optional.

Default


1, 0.5

Example

enabledragging=0

Swatches.iscommand

[Swatches.|<containerId>_swatches.]iscommand=*isCommand*

<i>isCommand</i>	<p>The Image Serving command string that is applied to all swatches. If it is specified in the URL, be sure that you HTTP-encode all occurrences of & and = as %26 and %3D, respectively.</p> <p> Note: Image sizing manipulation commands are not supported.</p>
------------------	---

Properties

Optional.

Default

None.

Example

When specified in the viewer URL:

iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000

When specified in the configuration data:

iscommand=op_sharpen=1&op_colorize=0xff0000

Swatches.maxloadradius

[Swatches.|<containerId>_swatches.]maxloadradius=-1 | 0 | *preloadnbr*

<code>-1 0 <i>preloadnbr</i></code>	<p>Specifies the component preload behavior. When set to <code>-1</code> all swatches will be loaded simultaneously when component is initialized or asset changed. When set to <code>0</code> only visible swatches are loaded.</p> <p><i>preloadnbr</i> defines how many invisible rows/columns around the visible area will be preloaded.</p>
---	--

Properties

Optional.

Default

1

Example

```
maxloadradius=-1
```

Swatches.pagemode

```
[Swatches.|<containerId>_swatches.]pagemode=0|1
```

<code>0 1</code>	<p>When toggled, the scroll buttons automatically cause the swatches to jump a full page length. Extra whitespace is shown on the last page if the swatches do not fit. Also, the last page has the same number of cells as any previous page.</p> <p>The scrollstep is ignored and mouse scrolling settles only on full pages.</p>
--------------------	---

Properties

Optional.

Default

0

Example

```
pagemode=1
```

Swatches.partialswatches

```
[Swatches.|<containerId>_swatches.]partialswatches=0|1
```

<code>0 1</code>	<p>Specifies whether the component allows scrolling to stop when any of the swatches are partially visible (scrolling is not aligned). The recommended value is <code>false</code> or <code>0</code>.</p>
--------------------	---

Properties

Optional.

Default

0

Example

```
partialswatches=1
```

Swatches.scrollstep

```
[Swatches. | <containerId>_swatches. ]scrollstep=hStep,vStep
```

<i>hStep</i>	Horizontal step.
<i>vStep</i>	Vertical step.

Specifies the number of swatches to scroll for each click or tap of the corresponding scroll button.

Properties

Optional.

Default

3, 3

Example

```
scrollstep=1,1
```

JavaScript API reference for Flyout Viewer

The main class of the Flyout Viewer is `FlyoutViewer`. It is declared in the `s7viewers` namespace. This JavaScript API covers constructor, methods, and call backs of this particular class.

In all the following examples, `<instance>` stands for the actual name of the JavaScript viewer object that is instantiated from the `s7viewers.FlyoutViewer` class.

dispose

JavaScript API reference for Flyout Viewer.

```
dispose()
```

Disposes this viewer instance by releasing all resources used by the viewer logic and deleting all inner objects and components created by the viewer in runtime.

The web page code should also delete the viewer instance variable as well to completely remove the viewer from the web browser memory.

If the web page code has registered event listeners directly on Viewer SDK components used by the viewer—or stored external references to such components—such listeners must be explicitly unregistered by the web page code, and such external component references must be deleted prior to calling `dispose()`.

Do not access the Viewer API any more after `dispose()` is called.

Parameters

None.

Returns

None.

Example

```
<instance>.dispose()
```

FlyoutViewer

JavaScript API reference for Flyout Viewer.

`FlyoutViewer([config])`

Constructor; creates a new Flyout Viewer instance.

Parameters

<i>config</i>	<p>{Object} optional JSON configuration object, allows all the viewer settings to pass to the constructor and avoid calling individual setter methods. Contains the following properties:</p> <ul style="list-style-type: none">• <code>containerId</code> – {String} ID of the DOM container (normally a <code>DIV</code>) that the viewer is inserted into. It is not necessary to have the container element created by the time this method is called. However, the container must exist when <code>init()</code> is run. Required.• <code>params</code> – {Object} JSON object with viewer configuration parameters where the property name is either a viewer-specific configuration option or an SDK modifier, and the value of that property is a corresponding settings value. Required.• <code>handlers</code> – {Object} JSON object with viewer event callbacks, where the property name is the name of supported viewer event, and the property value is a JavaScript function reference to an appropriate callback. Optional. See Event callbacks for more information about viewer events.• <code>localizedTexts</code> – {Object} JSON object with localization data. Optional. See Localization of user interface elements for more information. <p>See the <i>Viewer SDK User Guide</i> and the example for more information about the object's content. Optional.</p>
---------------	---

Returns

None.

Example

```
var flyoutViewer = new s7viewers.FlyoutViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  },
  "handlers": {
    "initComplete": function() {
      console.log("init complete");
    }
  },
  "localizedTexts": {
    "en": {
      "FlyoutZoomView.TIP_BUBBLE_OVER": "Mouse over to zoom"
    },
    "fr": {
      "FlyoutZoomView.TIP_BUBBLE_OVER": "Passez la souris sur pour zoomer"
    }
  },
  defaultLocale: "en"
});
```

getComponent

JavaScript API reference for Flyout Viewer

`getComponent(componentId)`

Returns a reference to the Viewer SDK component that is used by the viewer. The web page can use this method to extend or customize the behavior of the out-of-box viewer. Call this method only after the `initComplete` viewer callback has run; otherwise, the component may not be created yet by the viewer logic.

Parameters

componentID – {String} an ID of the Viewer SDK component used by the viewer. This viewer supports the following component IDs:

Component ID	Viewer SDK component class name
parameterManager	s7sdk.ParameterManager
container	s7sdk.common.Container
mediaSet	s7sdk.set.MediaSet
flyout	s7sdk.image.FlyoutZoomView
swatches	s7sdk.set.Swatches

When working with SDK APIs it is important to use a correct, fully qualified SDK namespace as described in [Viewer SDK](#). See the Viewer SDK API documentation for more information about a particular component.

Returns

{Object} a reference to Viewer SDK component. The method returns null if the `componentId` is not a supported viewer component or if the component was not yet created by the viewer logic.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    var flyoutZoomView = <instance>.getComponent("flyout");
  }
})
```

init

JavaScript API reference for Flyout Viewer.

`init()`

Starts the initialization of the Flyout Viewer. By this time the container DOM element must be created so that the viewer code can find it by its ID.

If the container element is not a part of the web page layout just yet (for example, it may be hidden using `display:none` style assigned to it), the viewer suspends its initialization process until the moment when the web page brings the container element back to the layout. When this occurs, the viewer load automatically resumes.

This method should be called only once during viewer life cycle, consequent calls are ignored.

Parameters

None.

Returns

{Object} A reference to the viewer instance.

Example

```
<instance>.init()
```

setAsset

JavaScript API reference for Flyouts Viewer.

`setAsset(asset)`

<i>asset</i>	<p>{String} new asset id, explicit image set or explicit image set with frame-specific Image Serving modifiers, with optional global Image Serving modifiers appended after ?.</p> <p>Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.</p>
--------------	---

Sets the new asset. You can call this parameter at any time, either before or after `init()`. If it is called after `init()`, the viewer swaps the asset at runtime.

See also [init](#).

Returns

None.

Example

Single image reference:

```
<instance>.setAsset("Scene7SharedAssets/Backpack_B")
```

Single reference to an image set that is defined in a catalog:

```
<instance>.setAsset("Scene7SharedAssets/ImageSet-Views-Sample")
```

Explicit image set:

```
<instance>.setAsset("Scene7SharedAssets/Backpack_B,Scene7SharedAssets/Backpack_C")
```

Explicit image set with frame-specific Image Serving modifiers:

```
<instance>.setAsset("(Scene7SharedAssets/Backpack_B?op_colorize=255%20%20,Scene7SharedAssets/Backpack_B?op_colorize=0x00ff00)")
```

Sharpening modifier added to all images in the set:

```
<instance>.setAsset("Scene7SharedAssets/Backpack_B,Scene7SharedAssets/Backpack_C?op_sharpen=1")
```

setContainerId

JavaScript API reference for Flyout Viewer.

```
setContainerId(containerId)
```

Sets the ID of the DOM container (normally a `DIV`) into which the viewer is inserted. It is not necessary to have the container element created by the time this method is called. However, the container must exist when `init()` is run. It must be called before `init()`. This method is optional if viewer configuration information is passed with `config` JSON object to the constructor.

<i>containerId</i>	{string} ID of container.
--------------------	---------------------------

Returns

None.

Example

```
<instance>.setContainerId("s7viewer");
```

setHandlers

JavaScript API reference for Flyout Viewer.

```
setHandlers(handlers)
```

Specifies zero or more callback handlers. A call to this method fully overwrites event handlers that were previously assigned for that viewer instance. Must be called before `init()`.

Parameter

<i>handlers</i>	<p>{Object} JSON object with viewer event callbacks, where the property name is the name of the supported viewer event and the property value is a JavaScript function reference to an appropriate callback.</p> <p>See Event callbacks for more information about viewer events.</p>
-----------------	---

Returns

None.

Example

```
<instance>.setHandlers({  
  "initComplete":function() {  
    console.log("init complete");  
  }  
})
```

setLocalizedTexts

JavaScript API reference for Flyout Viewer.

```
setLocalizedTexts(localizationInfo)
```

<i>localizationInfo</i>	<p>{Object} JSON object with localization data.</p> <p>See Localization of user interface elements for more information.</p> <p>See the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
-------------------------	---

Sets localization SYMBOL values for one or more locales. This parameter must be called before `init()`.

See also [init](#).

Returns

None.

Example

```
<instance>.setLocalizedTexts({ "en": { "FlyoutZoomView.TIP_BUBBLE_OVER": "Mouse over to zoom" }, "fr": { "FlyoutZoomView.TIP_BUBBLE_OVER": "Passez la souris sur pour zoomer" }, defaultLocale: "en" })
```

setParam

JavaScript API reference for Flyout Viewer.

```
setParam(name, value)
```

Sets the viewer parameter to a specified value. The parameter is either a viewer-specific configuration option or a software development kit modifier. This parameter is called before `init()`. This method is optional if viewer configuration information is passed with `config` JSON object to the constructor.

See also [init](#).

<i>name</i>	{string} name of parameter.
<i>value</i>	{string} value of parameter. The value cannot be percent-encoded.

Returns

None.

Example

```
<instance>.setParam( "style", "customStyle.css" )
```

setParams

JavaScript API reference for Flyout Viewer.

setParams(*params*)

<i>params</i>	{string} name=value parameter pairs separated with &.
---------------	---

Sets one or more parameters to a given value. The method argument syntax is identical to a URL query string. That is, it represents name=value pairs separated with &. The same as in a query string, names and values are percent-encoded using UTF8. Before you call `init()`, this parameter must be called. This method is optional if viewer configuration information is passed with `config` JSON object to the constructor.

See also [init](#).

Returns

None.

Example

```
<instance>.setParams( "FlyoutZoomView.zoomfactor=2,3&Swatches.iscommand=op_sharpen%3d1" )
```

Event callbacks

The viewer supports JavaScript event callbacks that the web page uses to track the viewer initialization process or runtime behavior.

Callback handlers are assigned by passing event names and corresponding handler functions with the `handlers` property to `config` JSON object in the viewer's constructor. Alternatively, it is possible to use `setHandlers()` API method.

Supported viewer events include the following:

- `initComplete` – triggers when viewer initialization is complete and all internal components are created, so that it is possible to use `getComponent()` API. The callback handler does not take any arguments.
- `trackEvent` – triggers each time an event occurs inside the viewer which may be handled by an event tracking system, such as Adobe Analytics. The callback handler takes the following arguments:
 - `objID` {String} not currently used.

- `compClass {String}` not currently used.
- `instName {String}` an instance name of the Viewer SDK component that triggered the event.
- `timeStamp {Number}` event time stamp.
- `eventInfo {String}` event payload.

See also [FlyoutViewer](#) and [setHandlers](#).

Customizing Flyout Viewer

All visual customization and most behavior customization is done by creating a custom CSS.

The suggested workflow is to take the default CSS file for the appropriate viewer, copy it to a different location, customize it, and specify the location of the customized file in the `style=` command.

Default CSS files can be found at the following location:

```
<s7_viewers_root>/html5/FlyoutViewer.css
```

Custom CSS file must contain the same class declarations as the default one. If a class declaration is omitted, the viewer does not function properly because it does not provide built-in default styles for user interface elements.

Alternative way to provide custom CSS rules is to use embedded styles directly on the web page or in one of linked external CSS rules.

When creating custom CSS keep in mind that the viewer assigns `.s7flyoutviewer` class to its container DOM element. If you are using external CSS file passed with `style=` command, use `.s7flyoutviewer` class as parent class in descendant selector for your CSS rules. If you are doing embedded styles on the web page, additionally qualify this selector with an ID of the container DOM element as follows:

```
#<containerId>.s7flyoutviewer
```

Building responsive designed CSS

It is possible to target different devices and embedding sizes in CSS to make your content display differently, depending on a user's device or a particular web page layout. This includes, but is not limited to, different web page layouts, user interface element sizes, and artwork resolution.

The viewer supports two methods for creating responsive designed CSS: CSS markers and standard CSS media queries. You can use these methods independently or together.

CSS markers

To assist in creating responsive designed CSS, the viewer supports CSS markers which special CSS classes dynamically assigned to the top-level viewer container element based on the run-time viewer size and the input type used on the current device.

The first group of CSS Markers includes `.s7size_large`, `.s7size_medium`, and `.s7size_small` classes. They are applied based on the runtime area of the viewer container. That is, if the viewer area is equal to or bigger than the size of a common desktop monitor `.s7size_large` is used; if the area is close in size to a common tablet device `.s7size_medium` is assigned. For areas similar to mobile phone screens `.s7size_small` is set. The primary purpose of these CSS markers is to create different user interface layouts for different screens and viewer sizes.

The second group of CSS Markers includes `.s7mouseinput` and `.s7touchinput`. `.s7touchinput` is set if the current device has touch input capabilities; otherwise, `.s7mouseinput` is used. These markers are intended to create user interface input elements with different screen sizes for different input types, because normally touch input requires larger elements. In cases

where the device has both mouse input and touch capabilities, `.s7touchinput` is set and the viewer renders a touch-friendly user interface.

The following sample CSS sets the zoom in button size to 28 x 28 pixels on systems with mouse input, and 56 x 56 pixels on touch devices. In addition, it hides the button completely if the viewer size becomes really small:

```
.s7flyoutviewer.s7mouseinput .s7swatches .s7thumb {
  width: 28px;
  height: 28px;
}
.s7flyoutviewer.s7touchinput .s7swatches .s7thumb {
  width: 56px;
  height: 56px;
}
.s7flyoutviewer.s7size_small .s7swatches {
  visibility: hidden;
}
```

To target devices with a different pixel density, use CSS media queries. The following media query block would contain CSS that is specific to high density screens:

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5)
{
}
```

Using CSS markers is the most flexible way of building responsive designed CSS that lets you to target not only device screen size but actual viewer size, which may be useful for responsive designed web page layouts.

CSS media queries

Device sensing can also be done using pure CSS media queries. Everything enclosed within a given media query block is applied only when it is run on a corresponding device.

When applied to Mobile Viewers, use four CSS media queries, defined in your CSS in the following order:

1. Contains only rules specific for all touch devices.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px)
{
}
```

2. Contains only rules specific for tablets with high resolution screens.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px) and (-webkit-min-device-pixel-ratio:1.5),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px) and (-webkit-min-device-pixel-ratio:1.5)
{
}
```

3. Contains only rules specific for all mobile phones.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in)
{
}
```

4. Contains only rules specific for mobile phones with high resolution screens.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in) and
(-webkit-min-device-pixel-ratio: 1.5),
only screen and (device-width:720px) and (device-height:1280px) and
(-webkit-device-pixel-ratio: 2),
only screen and (device-width:1280px) and (device-height:720px) and
(-webkit-device-pixel-ratio: 2)
```

```
{
}
```

Using a media queries approach, you should organize CSS with device sensing as follows:

- First, the desktop-specific section defines all properties that are either desktop-specific or common to all screens.
- And second, the four media queries go in the order defined above and provide CSS rules that are specific for the corresponding device type.

There is no need to duplicate the entire viewer CSS in each media query. Only properties that are specific to given devices are redefined inside a media query.

CSS Sprites

Many viewer user interface elements are styled using bitmap artwork and have more than one distinct visual state. A good example is a button that normally has at least 3 different states: "up", "over", and "down". Each state requires its own bitmap artwork assigned.

With a classic approach to styling, the CSS would have a separate reference to individual image file on the server for each state of the user interface element. The following is a sample CSS for styling scroll button:

```
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="up"]{
background-image:url(images/v2/ScrollLeftButton_up.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="over"]{
background-image:url(images/v2/ScrollLeftButton_over.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="down"]{
background-image:url(images/v2/ScrollLeftButton_down.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="disabled"]{
background-image:url(images/v2/ScrollLeftButton_disabled.png);
}
```

The drawback to this approach is that the end user experiences flickering or delayed user interface response when the element is interacted with for the first time. This action occurs because the image artwork for the new element state is not yet downloaded. Also, this approach may have a slight negative impact on performance because of an increase in the number of HTTP calls to the server.

CSS sprites is a different approach where image artwork for all element states is combined into a single PNG file called a "sprite". Such "sprite" has all visual states for the given element positioned one after another. When styling a user interface element with sprites the same sprite image is referenced for all different states in the CSS. Also, the `background-position` property is used for each state to specify which part of the "sprite" image is used. You can structure a "sprite" image in any suitable way. Viewers normally have it vertically stacked. Below is a "sprite"-based example of styling the same scroll button from above:

```
.s7flyoutviewer .s7scrollleftbutton[state] {
background-image: url(images/v2/ScrollLeftButton_light_sprite.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="up"]{
background-position: -56px -504px;
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="over"]{
background-position: -0px -504px;
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="down"]{
background-position: -56px -448px;
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="disabled"]{
background-position: -0px -448px;
}
```

General styling notes and advice

- When customizing the viewer user interface with CSS the use of the `!important` rule is not supported to style viewer elements. In particular, `!important` rule should not be used to override any default or run-time styling provided by the viewer or Viewer SDK. The reason is that it may affect the behavior of proper components. Instead, you should use CSS selectors with the proper specificity to set CSS properties that are documented in this reference guide.
- All paths to external assets within CSS are resolved against the CSS location, not the viewer HTML page location. Be aware of this rule when you copy the default CSS to a different location. Either copy the default assets as well or update paths within the custom CSS.
- The preferred format for bitmap artwork is PNG.
- Bitmap artwork is assigned to user interface elements using the `background-image` property.
- The `width` and `height` properties of a user interface element define its logical size. The size of the bitmap passed to `background-image` does not affect logical size.
- To use the high pixel density of high-resolution screens like Retina, specify bitmap artwork twice as large as the logical user interface element size. Then, apply the `-webkit-background-size:contain` property to scale the background down to the logical user interface element size.
- To remove a button from the user interface, add `display:none` to its CSS class.
- You can use various formats for color value that CSS supports. If you need transparency, use the format `rgba(R,G,B,A)`. Otherwise, you can use the format `#RRGGBB`.

Common User Interface Elements

The following is user interface elements reference documentation that applies to Flyout Viewer:

Flyout zoom view

The main view consists of the static image, the zoomed image shown in the flyout view, the highlight navigation area displayed over the static image and the tip message shown on top of static image.

If the dimensions of the image being viewed do not match the dimensions of the flyout zoom view, the image content is centered within the flyout zoom view's rectangle display area.

CSS properties of the main view

The appearance of the main view is controlled with the following CSS class selector:

```
.s7flyoutviewer .s7flyoutzoomview
```

CSS property	Description
<code>background-color</code>	The background color of the main view.

Example – to make the main view transparent:

```
.s7flyoutviewer .s7flyoutzoomview {
  background-color: transparent;
}
```

CSS properties of the flyout view

The appearance of the flyout view is controlled with the following CSS class selector:

```
.s7flyoutviewer .s7flyoutzoomview .s7flyoutzoom
```

CSS property	Description
left	The horizontal position of the flyout view, relative to top left corner of main view.
top	The vertical position of the flyout view, relative to top left corner of main view.
width	The width of the flyout view.
height	The height of the flyout view.
border	The border of the flyout view.

Example – to set up a flyout view to 600 x 400 pixels, appearing with 100 pixels offset to the right of the 512 x 288 main view shown in the previous example:

```
.s7flyoutviewer .s7flyoutzoomview .s7flyoutzoom {
  left: 612px;
  top: 0px;
  width: 600px;
  height: 400px;
}
```

CSS properties of the highlight in the main view

The appearance of the highlight in the main view is controlled with the following CSS class selector:

```
.s7flyoutviewer .s7flyoutzoomview .s7highlight
```

It is possible to control background, border, transparency and similar attributes using CSS. However, the size and position of highlight DOM element is managed by the viewer logic. Overriding it through CSS is not supported.

CSS property	Description
background-color	The color of the highlight.
opacity	Highlight opacity. For Internet Explorer 8, use <code>filter:alpha(opacity=...) </code> ;
border	The border highlight.

Example – to set up green highlight with 40% transparency and a one pixel red border:

```
.s7flyoutviewer .s7flyoutzoomview .s7highlight {
  background-color: green;
  opacity: 0.4;
  filter: alpha(opacity = 40);
  border: 1px solid red;
}
```

CSS properties of the cursor

When `highlightmode` parameter is set to `cursor`, highlight are in the main view is replaced with fixed-size cursor artwork, which is controlled with the CSS class selector:

```
.s7flyoutviewer .s7flyoutzoomview  
.s7cursor
```

It is possible to control the background image and size using CSS.

Applicable CSS properties include:

CSS property	Description
<code>background-image</code>	Cursor artwork.
<code>width</code>	Cursor width.
<code>height</code>	Cursor height.



Note: Cursor supports the `input` attribute selector, which can be used to apply different cursor artwork and size for different devices. In particular, `input="mouse"` corresponds to the desktop systems and `input="touch"` corresponds to the touch devices.

CSS properties of the overlay

When the `overlay` parameter is set to 1, the area around the highlight frame or the cursor image is controlled with the CSS class selector:

```
.s7flyoutviewer .s7flyoutzoomview  
.s7overlay
```

CSS property	Description
<code>background-color</code>	Overlay color.
<code>opacity</code>	Overlay opacity.

CSS properties of the tip message

The appearance of the tip message is controlled with the following CSS class selector:

```
.s7flyoutviewer .s7flyoutzoomview .s7tip
```

It is possible to configure font style, size appearance and vertical offset through CSS. However, horizontal alignment is managed by the viewer logic. Overriding it through CSS using `left` or `right` properties is not supported.

CSS property	Description
<code>bottom</code>	Offset from the bottom of the main view.
<code>color</code>	Text color.

CSS property	Description
font-family	Font name.
font-size	Font size.
padding	Padding around the message text.
background-color	Background fill color of message text.
border-radius	Background border radius of message text.
opacity	Background opacity of message text. For Internet Explorer 8, use <code>filter:alpha(opacity=...)</code>)

The tip message can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up semi-transparent tip message with white Arial 12px font, 50 pixels offset from the bottom of the main view, padding, and a rounded border:

```
.s7flyoutviewer .s7flyoutzoomview .s7tip {
bottom: 50px;
color: #ffffff;
font-family: Arial;
font-size: 12px;
padding-bottom: 10px;
padding-top: 10px;
padding-left: 12px;
padding-right: 12px;
background-color: #000000;
border-radius: 4px;
opacity: 0.5;
filter: alpha(opacity = 50);
}
```

Focus highlight

Input focus highlight displayed around focused viewer UI element is controlled with the CSS class selector.

CSS properties

The appearance is controlled with the following CSS class selector:

```
.s7flyoutviewer *:focus
```

CSS property	Description
outline	Focus highlight style.

Example – to disable the default browser focus highlight for all viewer user interface elements, add the following CSS selector to viewer's style sheet:

```
.s7flyoutviewer *:focus {
  outline: none;
}
```

Main viewer area

The main view area is the area occupied by the flyout view and swatches.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7flyoutviewer
```

CSS property	Description
width	The width of the viewer.
height	The height of the viewer.
background-color	Background color in hexadecimal format.

Example – to set up a flyout viewer with white background (#FFFFFF) and make its size 260 x 500 pixels.

```
.s7flyoutviewer {
  background-color: #FFFFFF;
  width: 260px;
  height: 500px;
}
```

Swatches

Swatches consist of a row of thumbnail images with optional scroll buttons on the left and right hand side.

Scroll buttons are only visible on the desktop if all thumbnails cannot fit into the width of the container. On mobile devices, or if thumbnails can fit into the container width, scroll buttons are not shown.

CSS properties of the swatches

The appearance of the swatches container is controlled with the following CSS class selector:

```
.s7flyoutviewer .s7swatches
```

CSS property	Description
width	The width of the swatches.
height	The height of the swatches.
bottom	The vertical swatches offset relative to the viewer container.

Example – to set up swatches to 460 x 100 pixels:

```
.s7flyoutviewer .s7swatches {
  width: 460px;
  height: 100px;
}
```

CSS properties of the thumbnail swatch spacing

The spacing between swatch thumbnails is controlled with the CSS class selector:

```
.s7flyoutviewer .s7swatches .s7thumbcell
```

CSS property	Description
margin	The size of the horizontal and vertical margin around each thumbnail. Actual thumbnail spacing equals to the sum of left and right margin set for <code>.s7thumbcell</code> .

Example – to set up spacing to be 10 pixels both vertically and horizontally:

```
.s7flyoutviewer .s7swatches .s7thumbcell {
  margin: 5px;
}
```

CSS properties of the thumbnail swatches

The appearance of individual thumbnail is controlled with the following CSS class selector:

```
.s7flyoutviewer .s7swatches .s7thumb
```

CSS property	Description
width	The width of the thumbnail swatches.
height	The height of the thumbnail swatches.
border	The border of the thumbnail swatches.



Note: Thumbnail supports the `state` attribute selector, which is used to apply different skins to different thumbnail states. In particular, `state="selected"` corresponds to the thumbnail for the image that is currently displayed in the main view, `state="default"` corresponds to the rest of the thumbnails, and `state="over"` is used on mouse hover.

Example – to set up thumbnails that are 56 x 56 pixels, have a light grey default border, and a dark grey selected border:

```
.s7flyoutviewer .s7swatches .s7thumb {
  width: 56px;
  height: 56px;
}
.s7flyoutviewer .s7swatches .s7thumb[state="default"] {
  border: 1px solid #dddddd;
}
.s7flyoutviewer .s7swatches .s7thumb[state="selected"] {
  border: 1px solid #666666;
}
```

CSS properties of the left and right scroll buttons

The appearance of left and right scroll buttons are controlled with the following CSS class selectors:

```
.s7flyoutviewer .s7swatches .s7scrollleftbutton
.s7flyoutviewer .s7swatches .s7scrollrightbutton
```

It is not possible to position scroll buttons using CSS top, left, bottom, and right properties. Instead, the viewer logic positions them automatically.

CSS property	Description
width	The width of the scroll button.
height	The height of the scroll button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the *state* attribute selector, which is used to apply different skins to button states up, down, over, and disabled.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up scroll buttons that are 56 x 56 pixels and have different artwork for each state:

```
.s7flyoutviewer .s7swatches .s7scrollleftbutton {
  background-size: auto;
  width: 56px;
  height: 56px;
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="up"]{
background-image:url(images/v2/ScrollLeftButton_up.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="over"]{
  background-image:url(images/v2/ScrollLeftButton_over.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="down"]{
  background-image:url(images/v2/ScrollLeftButton_down.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="disabled"]{
  background-image:url(images/v2/ScrollLeftButton_disabled.png);
}
.s7flyoutviewer .s7swatches .s7scrollrightbutton {
  background-size: auto;
  width: 56px;
  height: 56px;
}
.s7flyoutviewer .s7swatches .s7scrollrightbutton[state="up"]{
background-image:url(images/v2/ScrollRightButton_up.png);
}
.s7flyoutviewer .s7swatches .s7scrollrightbutton[state="over"]{
  background-image:url(images/v2/ScrollRightButton_over.png);
}
.s7flyoutviewer .s7swatches .s7scrollrightbutton[state="down"]{
  background-image:url(images/v2/ScrollRightButton_down.png);
}
.s7flyoutviewer .s7swatches .s7scrollrightbutton[state="disabled"]{
```

```
background-image:url(images/v2/ScrollRightButton_disabled.png);
}
```

Tooltips

On desktop systems some user interface elements like buttons have tooltips that are displayed on mouse hover.

CSS properties of the main viewer area

The appearance of tooltips is controlled with the following CSS class selector:

```
.s7tooltip
```

CSS property	Description
border-radius	Background border radius.
border-color	Background border color.
background-color	Background color.
color	Text color.
font-family	Text font name.
font-size	Text font size.



Note: In case tooltip styles are customized from within the embedding web page, all properties have to contain **!IMPORTANT** rule. This is not necessary if tooltips are customized in the viewer's CSS file.

Example – to set up tooltips that have a grey border with 3px corner radius, black background and white text written with Arial, 11 pixels size:

```
.s7tooltip {
border-radius: 3px 3px 3px 3px;
border-color: #999999;
background-color: #000000;
color: #FFFFFF;
font-family: Arial, Helvetica, sans-serif;
font-size: 11px;
}
```

Support for Adobe Analytics tracking

The Flyout Viewer supports Adobe Analytics tracking out of the box.

Out-of-the-box tracking

The Flyout Viewer supports Adobe Analytics tracking out-of-the-box. To enable tracking, pass the proper company preset name as `config2` parameter.

The viewer also sends a single tracking HTTP request to the configured Image Server with the viewer type and version information.

Custom tracking

To integrate with third-party analytics systems it is necessary to listen to the `trackEvent` viewer callback and process the `eventInfo` argument of the callback function as necessary. The following code is an example of such handler function:

```
var flyoutViewer = new s7viewers.FlyoutViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  },
  "handlers": {
    "trackEvent": function(objID, compClass, instName, timeStamp, eventInfo) {
      //identify event type
      var eventType = eventInfo.split(",")[0];
      switch (eventType) {
        case "LOAD":
          //custom event processing code
          break;
          //additional cases for other events
      }
    }
  }
});
```

The viewer tracks the following SDK user events:

SDK user event	Sent when...
LOAD	the viewer is loaded first.
SWAP	an asset is swapped in the viewer using <code>setAsset()</code> API.
ZOOM	the flyout is activated or the zoom level is changed.
PAN	an image is panned.
SWATCH	an image is changed by clicking or tapping on a swatch.

Localization of user interface elements

Certain content that the Flyout Viewer displays is subject to localization. This content includes user interface element tool tips and information messages that are displayed by the flyout zoom view on load.

Every textual content in the viewer that can be localized is represented by a special Viewer SDK identifier called SYMBOL. Any SYMBOL has a default associated text value for the English locale ("en") supplied with the out-of-the-box viewer. It also may have user-defined values set for as many locales as needed.

When the viewer starts, it checks the current locale to see if there is a user-defined value for each supported SYMBOL for the locale. If there is, it uses the user-defined value; otherwise, it falls back to the out-of-the-box default text.

User-defined localization data can be passed to the viewer as a localization JSON object. Such object contains the list of supported locales, SYMBOL text values for each locale, and the default locale.

An example of such a localization object is the following:

```
{
  "en": {
    "FlyoutZoomView.TIP_BUBBLE_OVER": "Mouse over to zoom",
    "FlyoutZoomView.TIP_BUBBLE_TAP": "Tap and hold to zoom"
  },
  "fr": {
    "FlyoutZoomView.TIP_BUBBLE_OVER": "Passez la souris sur pour zoomer",
    "FlyoutZoomView.TIP_BUBBLE_TAP": "Appuyez et maintenez pour agrandir"
  },
  defaultLocale: "en"
}
```

In the above example, the localization object defines two locales ("en" and "fr") and provides localization for two user interface elements in each locale.

The web page code should pass the localization object to the viewer constructor, as a value of the `localizedTexts` field of the configuration object. An alternative option is to pass the localization object by calling the `setLocalizedTexts(localizationInfo)` method.

The following SYMBOLs are supported:

SYMBOL	Description
<code>FlyoutZoomView.TIP_BUBBLE_OVER</code>	Information message for desktop systems.
<code>FlyoutZoomView.TIP_BUBBLE_TAP</code>	Information message for touch devices.
<code>ScrollLeftButton.TOOLTIP</code>	Tooltip for scroll left button.
<code>ScrollRightButton.TOOLTIP</code>	Tooltip for scroll right button.
<code>ScrollUpButton.TOOLTIP</code>	Tooltip for scroll up button.
<code>ScrollDownButton.TOOLTIP</code>	Tooltip for scroll down button.

Viewer SDK namespace

The viewer is built of many Viewer SDK components. In most cases, the web page does not need to interact with SDK components API directly; all common needs are covered in the viewer API itself.

However, some advanced use cases require that the web page obtain a reference to an inner SDK component using the `getComponent()` viewer API and then use all the flexibility of the APIs of SDK itself.

The namespace that is used to load and initialize SDK components by the viewer depends on the environment in which the viewer is operating. If the viewer is running in AEM (Adobe Experience Manager), the viewer loads SDK components into `s7viewers.s7sdk` namespace. Likewise, the viewer served from Scene7 Publishing System loads the SDK into `s7classic.s7sdk`.

In either case, the namespace used by the SDK inside the viewer has either `s7viewers` or `s7classic` as the prefix. And, it is different from the plain `s7sdk` namespace used in the SDK User Guide or SDK API documentation. For that reason, it is important to use a fully qualified SDK namespace when you write custom application code that communicates with internal viewer components.

For example, if you plan to listen to `StatusEvent.NOTF_VIEW_READY` event and the viewer is served from AEM, the fully qualified event type is `s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY`, and the event listener code looks similar to the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var flyout = <instance>.getComponent("flyout");
    flyout.addEventListener(s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});
```

The same code for viewer served from Scene7 SPS will look like this:

```
<instance>.setHandlers({
  "initComplete":function() {
    var flyout = <instance>.getComponent("flyout");
    flyout.addEventListener(s7classic.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});
```

Inline Zoom

Inline Zoom Viewer is an image viewer. It displays a static image with the zoomed version shown over that static image when a user rolls over or touches the main view. This viewer works with image sets and navigation is done by using swatches. It is designed to work on desktops and mobile devices.



Note: Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.

Viewer type is 504.

See [System requirements](#).

Demo URL

http://s7.scene7.com/s7viewers/html5/flyoutViewer.html?asset=Scene7SharedAssets/ImageSet/Views/Sample/config=Scene7SharedAssets/Universal_HTML5_Zoom_Inline&stageSize=500,400

Using Inline Zoom Viewer

Inline Zoom Viewer represents a main JavaScript file and a set of helper files (a single JavaScript include with all Viewer SDK components used by this particular viewer, assets, CSS) downloaded by the viewer at runtime.

Inline Zoom Viewer can be used both in pop-up mode using production-ready HTML page provided with Image Serving Viewers or in embedded mode where it is integrated into a target web page using documented API.

Configuration and skinning are similar to that of the other viewers. You can use custom CSS to apply skinning.

See [Command reference common to all viewers – Configuration attributes](#) and [Command reference common to all Viewers – URL](#)

Interacting with Inline Zoom Viewer

Inline Zoom Viewer supports single-touch and multi-touch gestures that are common in other mobile applications.

Gesture	Description
Single tap	Activates the flyout view or changes between the primary and secondary zoom level in swatches, selects new thumbnail.
Horizontal swipe or flick	Scrolls through the list of swatches in the swatch bar.
Vertical swipe	If the gesture is done within the swatches area, it performs a native page scroll.

The viewer also supports both touch input and mouse input on Windows devices with touch screen and mouse. This support, however, is limited to Chrome, Internet Explorer 11, and Edge web browsers only.

This viewer is fully keyboard accessible.

See [Keyboard accessibility and navigation](#).

Embedding Inline Zoom Viewer

Different web pages have different needs for viewer behavior. Sometimes a web page provides a clickable link that opens the viewer in a separate browser window. In other cases, it may be necessary to embed the viewer directly in the hosting page. In the latter case, the web page may have static page layout, or use responsive design that displays differently on different devices, or for different browser window sizes. To accommodate these needs, the viewer supports three primary operation modes: pop-up, fixed size embedding, and responsive embedding.

Pop-up

In the pop-up mode the viewer is opened in a separate web browser window or tab. It takes the entire browser window area and adjusts when the browser window is resized or the device orientation is changed.

This mode is the most common for mobile devices. The web page loads the viewer using `window.open()` JavaScript call, properly configured A HTML element or any other suitable way.

It is recommended that you use the out-of-the-box HTML page for pop-up mode called `FlyoutViewer.html`. It is located under the `html5/` subfolder of your standard Image Serving-Viewers deployment:

```
<s7viewers_root>/html5/FlyoutViewer.html
```

It is also necessary to have the `FlyoutZoomView` component configured to work in the inline zoom mode. It is recommended that you use the out-of-the-box `Scene7SharedAssets/Universal_HTML5_Zoom_Inline` preset for the Inline Zoom viewer, or a custom preset derived from it. Visual customization can be achieved by applying custom CSS.

The following is an HTML code example that opens the viewer in a new window:

```
<a
href="http://s7l.scene7.com/s7viewers/html5/FlyoutViewer.html?asset=Scene7SharedAssets/ImageSt-VueSampleConfig/Scene7SharedAssets/Universal_HTML5_Zoom_Inline"target="_blank"open
popup viewer">
```

Fixed Size Embedding and Responsive Embedding

In the embedded mode the viewer is added to the existing web page, which may already have some customer content not related to the viewer. The viewer normally occupies only a part of web page real estate.

The primary use case are web pages oriented for desktops or tablet devices, and also responsive web pages which adjust layout automatically depending on the device type.

Fixed size embedding mode is used when the viewer does not change its size after its initial load. This choice is best for web pages that have a static page layout.

Responsive design embedding mode assumes that the viewer may need to resize during runtime in response to the size change of its container `DIV`. The most common use case is adding a viewer to a web page that uses a flexible page layout.

When using responsive design embedding mode with the Inline Zoom Viewer, make sure that you specify explicit breakpoints for the main view image using the `imageload` parameter. Ideally, match your breakpoints with the viewer width breakpoints as dictated by the web page CSS.

In responsive design embedding mode the viewer behaves differently depending on the way a web page sizes its container `DIV`. If the web page sets only the width of the container `DIV`, leaving its height unrestricted, then the viewer automatically chooses its height according to the aspect ratio of the asset that is used. This means that the asset fits perfectly into the view without any padding on the sides. This particular use case is the most common for web pages that use responsive design layout frameworks like Bootstrap, Foundation, and so forth.

Otherwise, if the web page sets both the width and the height for the viewer's container `DIV`, then the viewer fills only that area and follows the size provided by the web page layout. A good use case example is embedding the viewer into a modal overlay, where the overlay is sized according to web browser window size.

Fixed size embedding

You add the viewer to a web page by doing the following:

1. Adding the viewer JavaScript file to your web page.
 2. Defining the container `DIV`.
 3. Setting the viewer size.
 4. Creating and initializing the viewer.
1. Adding the viewer JavaScript file to your web page.

Creating a viewer requires that you add a script tag in the HTML head. Before you can use the viewer API, be sure that you include `FlyoutViewer.js`. `FlyoutViewer.js` is located in the following `html5/js/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/js/FlyoutViewer.js
```

You can use a relative path if the viewer is deployed on one of the Adobe Scene7 servers and it is served from the same domain. Otherwise, you specify a full path to one of Adobe Scene7 servers that have the IS-Viewers installed.

A relative path looks like the following:

```
<script language="javascript" type="text/javascript"
src="/s7viewers/html5/js/FlyoutViewer.js"></script>
```



Note: You should only reference the main viewer JavaScript `include` file on your page. You should not reference any additional JavaScript files in the web page code which might be downloaded by the viewer's logic in runtime. In particular, do not directly reference `HTML5 SDK Utils.js` library loaded by the viewer from `/s7viewers` context path (so-called consolidated SDK `include`). The reason is that the location of `Utils.js` or similar runtime viewer libraries is fully managed by the viewer's logic and the location changes between viewer releases. Adobe does not keep older versions of secondary viewer `includes` on the server.

As a result, putting a direct reference to any secondary JavaScript `include` used by the viewer on the page breaks the viewer functionality in the future when a new product version is deployed.

2. Defining the container DIV.

Add an empty DIV element to the page where you want the viewer to appear. The DIV element must have its ID defined because this ID is later passed to the viewer API.

The placeholder DIV is a positioned element, meaning that the `position` CSS property is set to `relative` or `absolute`.

It is the responsibility of the web page to specify the proper `z-index` for the placeholder DIV element. Doing so ensures that the viewer's flyout portion appears on top of the other web page elements.

The following is an example of a defined placeholder DIV element:

```
<div id="s7viewer" style="position:relative;z-index:1"></div>
```

3. Setting the viewer size.

This viewer displays thumbnails when working with multi-item sets. On desktop systems, thumbnails are placed below the main view. At the same time, the viewer allows the swapping of the main asset during runtime using `setAsset()` API. As a developer, you have control over how the viewer manages the thumbnails area in the bottom area when the new asset has only one item. It is possible to keep the outer viewer size intact and let the main view increase its height and occupy the thumbnails area. Or, you can keep the main view size static and collapse the outer viewer area, thereby letting web page content to move up, and then use the free page real estate left from the thumbnails.

To keep the outer viewer bounds intact define the size for `.s7flyoutviewer` top-level CSS class in absolute units. Sizing in CSS can be put right on the HTML page, or in a custom viewer CSS file, which is later assigned to a viewer preset record in Scene7 Publishing System, or passed explicitly using `style` command.

See [Customizing Inline Zoom Viewer](#) for more information about styling the viewer with CSS.

The following is an example of defining the static outer viewer size in an HTML page:

```
#s7viewer.s7flyoutviewer {
  width: 640px;
  height: 480px;
}
```

You can see the behavior with a fixed outer viewer area on the following sample page. Notice that when you switch between sets, the outer viewer size does not change:

https://marketing.adobe.com/resources/help/en_US/s7/viewers_ref/samples/InlineZoom-fixed-outer-area.html

To make the main view dimensions static, define the viewer size in absolute units for the inner `Container` SDK component using `.s7flyoutviewer .s7container` CSS selector. In addition, you should override the fixed size defined for the `.s7flyoutviewer` top-level CSS class in the default viewer CSS, by setting it to `auto`.

The following is an example of defining the viewer size for the inner `Container` SDK component so that the main view area does not change its size when switching the asset:

```
#s7viewer.s7flyoutviewer {
  width: auto;
  height: auto;
}
#s7viewer.s7flyoutviewer .s7container {
  width: 640px;
  height: 480px;
}
```

The following sample page shows viewer behavior with a fixed main view size. Notice that when you switch between sets, the main view remains static and the web page content moves vertically:

https://marketing.adobe.com/resources/help/en_US/s7/viewers_ref/samples/InlineZoom-fixed-main-view.html

Also, note that the default viewer CSS provides a fixed size for its outer area out-of-the-box.

4. Creating and initializing the viewer.

When you have completed the steps above, you create an instance of `s7viewers.FlyoutViewer` class, pass all configuration information to its constructor and call `init()` method on a viewer instance. Configuration information is passed to the constructor as a JSON object. At minimum, this object should have the `containerId` field which holds the name of viewer container ID and nested `params` JSON object with configuration parameters that the viewer supports. In this case, the `params` object must have at least the Image Serving URL passed as `serverUrl` property, the initial asset as `asset` parameter, base path for loading CSS as `contentUrl` parameter, and preset name as `config` parameter.. JSON-based initialization API lets you create and start the viewer with single line of code.

It is important to have the viewer container added to the DOM so that the viewer code can find the container element by its ID. Some browsers delay building DOM until the end of the web page. For maximum compatibility, call the `init()` method just before the closing BODY tag, or on the `body onload()` event.

At the same time, the container element should not necessarily be part of the web page layout just yet. For example, it may be hidden using `display:none` style assigned to it. In this case, the viewer delays its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load automatically resumes.

The following is an example of creating a viewer instance, passing the minimum necessary configuration options to the constructor and calling the `init()` method. The example assumes `inlineZoomViewer` is the viewer instance; `s7viewer` is the name of placeholder DIV; `http://s7d1.scene7.com/is/image/` is the Image Serving URL; and `Scene7SharedAssets/ImageSet-Views-Sample` is the asset:

```
<script type="text/javascript">
var inlineZoomViewer = new s7viewers.FlyoutViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
    "config": "Scene7SharedAssets/Universal_HTML5_Zoom_Inline",
    "contenturl": "http://s7d1.scene7.com/is/content/",
    "serverurl": "http://s7d1.scene7.com/is/image/"
  }
}).init();
</script>
```

The following code is a complete example of a trivial web page that embeds the Inline Zoom Viewer with a fixed size:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7d1.scene7.com/s7viewers/html5/js/FlyoutViewer.js"></script>
<style type="text/css">
#s7viewer.s7flyoutviewer {
  width: 640px;
  height: 480px;
}
</style>
</head><body>
<div id="s7viewer" style="position:relative;z-index:1;"></div>
<script type="text/javascript">
var inlineZoomViewer = new s7viewers.FlyoutViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
    "config": "Scene7SharedAssets/Universal_HTML5_Zoom_Inline",
    "contenturl": "http://s7d1.scene7.com/is/content/",
    "serverurl": "http://s7d1.scene7.com/is/image/"
  }
}
```

```

    }).init();
</script>
</body>
</html>

```

Responsive design embedding with unrestricted height

With responsive design embedding, the web page normally has some kind of flexible layout in place that dictates the runtime size of the viewer's container DIV. For the following example, assume that the web page allows the viewer's container DIV to take 40% of the web browser window size, leaving its height unrestricted. The web page HTML code would look like the following:

```

<!DOCTYPE html>
<html>
<head>
<style type="text/css">
.holder {
width: 40%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>

```

Adding the viewer to such a page is similar to the steps for fixed size embedding. The only difference is that you need to override the fixed sizing from the default viewer CSS with the size set in relative units.

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Setting the viewer size.
4. Creating and initializing the viewer.

All the steps above are the same as with the fixed size embedding with the following three exceptions:

- add the container DIV to the existing "holder" DIV;
- added `imagereload` parameter with explicit breakpoints;
- instead of setting a fixed viewer size using absolute units use CSS that sets the viewer width and height to 100% as in the following:

```

#s7viewer.s7flyoutviewer {
width: 100%;
height: 100%;
}

```

The following code is a complete example. Notice how the viewer size changes when the browser is resized, and how the viewer aspect ratio matches the asset.

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/FlyoutViewer.js"></script>
<style type="text/css">
.holder {
width: 40%;
}
#s7viewer.s7flyoutviewer {
width: 100%;
height: 100%;
}

```

```

}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative;z-index:1"></div>
</div>
<script type="text/javascript">
var inlineZoomViewer = new s7viewers.FlyoutViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Scene7SharedAssets/ImageSet-Views-Sample",
    "config": "Scene7SharedAssets/Universal_HTML5_Zoom_Inline",
    "contenturl": "http://s7dl.scene7.com/is/content/",
    "serverurl":"http://s7dl.scene7.com/is/image/",
    "imagereload":"1,breakpoint,200;400;800;1600"
  }
}).init();
</script>
</body>
</html>

```

The following examples page illustrates more real-life uses of responsive design embedding with unrestricted height:

https://marketing.adobe.com/resources/help/en_US/s7/vlist/vlist.html

Flexible size embedding with width and height defined

In case of flexible-size embedding with width and height defined, the web page styling is different. It provides both sizes to the "holder" DIV and centers it in the browser window. Also, the web page sets the size of the HTML and BODY element to 100 percent.

```

<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>

```

The rest of the embedding steps are identical to the steps used for responsive design embedding with unrestricted height. The resulting example is the following:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/FlyoutViewer.js"></script>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}

```

```

.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
#s7viewer.s7flyoutviewer {
  width: 100%;
  height: 100%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative;z-index:1"></div>
</div>
<script type="text/javascript">
var inlineZoomViewer = new s7viewers.FlyoutViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
    "config": "Scene7SharedAssets/Universal_HTML5_Zoom_Inline",
    "contenturl": "http://s7dl.scene7.com/is/content/",
    "serverurl": "http://s7dl.scene7.com/is/image/",
    "imagereload": "1,breakpoint,200;400;800;1600"
  }
}).init();
</script>
</body>
</html>

```

Embedding using Setter-based API

Instead of using JSON-based initialization, it is possible to use setter-based API and no-args constructor. Using this API constructor does not take any parameters and configuration parameters are specified using `setContainerId()`, `setParam()`, and `setAsset()` API methods, with separate JavaScript calls.

The following example illustrates using fixed size embedding with setter-based API:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/FlyoutViewer.js"></script>
<style type="text/css">
#s7viewer.s7flyoutviewer {
  width: 640px;
  height: 480px;
}
</style>
</head><body>
<div id="s7viewer" style="position:relative;z-index:1;"></div>
<script type="text/javascript">
var inlineZoomViewer = new s7viewers.FlyoutViewer();
inlineZoomViewer.setContainerId("s7viewer");
inlineZoomViewer.setParam("serverurl", "http://s7dl.scene7.com/is/image/");
inlineZoomViewer.setParam("config", "Scene7SharedAssets/Universal_HTML5_Zoom_Inline");
inlineZoomViewer.setParam("contenturl", "http://s7dl.scene7.com/is/content/");
inlineZoomViewer.setAsset("Scene7SharedAssets/ImageSet-Views-Sample");
inlineZoomViewer.init();
</script>
</body>
</html>

```

Command reference – Configuration attributes

Configuration attributes documentation for Flyout Viewer

You can set any configuration command in URL. Or, you can use `setParam()`, `setParams()`, or both API methods. You can also specify any configuration attribute in the server-side configuration record.

Some configuration commands are prefixed with the class name or instance name of corresponding Viewer SDK component. An instance name of the component is dynamic and depends on the ID of the viewer container DOM element passed to `setContainerId()` API method. Documentation includes an optional prefix for such commands. For example, the `zoomfactor` command is documented as follows:

```
[FlyoutZoomView.|<containerId>_flyout].zoomfactor
```

The command is used as follows:

- `zoomfactor` (short syntax)
- `FlyoutZoomView.zoomfactor` (qualified with a component class name)
- `cont_flyout.zoomfactor` (qualified with the component ID, assuming that `cont` is the ID of the container element)

See also [Command reference common to all viewers – Configuration attributes](#)

FlyoutZoomView.flyouttransition

```
[FlyoutZoomView.|<containerId>_flyout.]flyouttransition=[none|slide|fade][,showtime[,showdelay[,hidetime[,hidedelay]]]]
```

<i>none/slide/fade</i>	Specifies the type the effect applied when the flyout view is shown or hidden. With <code>none</code> , the flyout image appears instantly when activated and ready; <code>slide</code> makes the slide animation play in left-to-right direction; <code>fade</code> applies an alpha transition to the flyout image.
<i>showtime</i>	Number of seconds that the show animation takes to complete.
<i>showdelay</i>	The delay in seconds between user action which initiates the show animation and the beginning of show animation itself.
<i>hidetime</i>	Number of seconds that the hide animation takes to complete.
<i>hidedelay</i>	The delay in seconds between user action which initiates the hide animation and the beginning of hide animation itself.

Properties

Optional.

Default

`fade,1,0,1,0`

Example

`flyouttransition=slide,1,1,2,1`

FlyoutZoomView.fmt

`[FlyoutZoomView. | <containerId>_flyout.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha`

<code>jpg jpeg png png-alpha gif gif-alpha</code>	<p>Specifies image format to be used by the component for loading images from Image Server. If the specified format ends with <code>-alpha</code>, the component renders images as transparent content. For all other image formats the component treats images as opaque.</p> <p>Note that the component has a white background by default. Therefore, to make it completely transparent set the <code>background-color</code> CSS property to <code>transparent</code>.</p>
---	---

Properties

Optional.

Default

`jpeg`

Example

`fmt=png-alpha`

FlyoutZoomView.frametransition

`[FlyoutZoomView. | <containerId>_flyout.]frametransition=none|fade[,duration]`

<code>none fade</code>	<p>Specifies the type of the effect applied to main view on asset change.</p> <p><code>none</code> stands for no transition, main view change happens instantly.</p> <p><code>fade</code> activates cross-fade transition where the old image fades out and the new image fades in</p>
<code>duration</code>	<p>Number of seconds that the animation takes to complete.</p>

Properties

Optional.

Default

None.

Example

`frametransition=fade,1`

FlyoutZoomView.imagereload

[FlyoutZoomView. | <containerId>_flyout.]imagereload=0|1[,breakpoint,width[:width]]

0 1	<p>Configures how the component fetches new images for the main and flyout view during resize.</p> <p>Set to 0, the component does not load new images during resize, and image resolution in the flyout view does not change.</p> <p>Set to 1 lets you specify one or more width breakpoints for the image loaded into the main view.</p>
breakpoint,width[:width]	<p>Width breakpoints for the image that is loaded into the main view.</p> <p>The component always uses the best fit size for the initial load. After resize it ensures that the image in the main view is always downloaded using the width that is equal to the closest larger breakpoint, and downscaled on the client.</p>

Properties

Optional.

Default


1,breakpoint,300;600;1200

Example

imagereload=1,breakpoint,200;400;800;1600

FlyoutZoomView.iscommand

[FlyoutZoomView. | <containerId>_flyout.]iscommand=isCommand

isCommand	<p>The Image Serving command string that is applied to the FlyoutZoomView main image and the zoomed in view. If it is specified in the URL, be sure that you HTTP-encode all occurrences of & and = as %26 and %3D, respectively.</p> <p> Note: Image sizing manipulation commands are not supported.</p>
-----------	---

Properties

Optional.

Default

None.

Example

When specified in the viewer URL:


```
iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000
```

When specified in the configuration data:

```
iscommand=op_sharpen=1&op_colorize=0xff0000
```

FlyoutZoomView.preloadtiles

```
[FlyoutZoomView. | <containerId>_flyout. ]preloadtiles=0|1
```

0 1	<p>Set to 1 to enable preloading of the zoomed image, or set to 0 to load the zoom image incrementally, as needed.</p> <p> Note: If you enable this option it can result in substantially higher bandwidth usage. The zoomed image is loaded in its entirety, even if the user does not initiate a zoom action.</p>
-----	---

Properties

Optional.

Default

0

Example

```
preloadtiles=1
```

FlyoutZoomView.tip

```
[FlyoutZoomView. | <containerId>_flyout. ]tip=duration[,count][,fade]
```

<i>duration</i>	Specifies the number of seconds the tip text is displayed before it hides. When set to -1, the message is always displayed, even if the user activates the flyout.
<i>count</i>	Specifies the number of times the text is displayed when viewing new images in the set. A value of -1 means that the text is always displayed when viewing any image in the set.
<i>fade</i>	Specifies the duration of a fade animation that occurs when the text appears or disappears. A value of 0 means no fade transition.

Properties

Optional.

Default

3,1,0.3

Example

tip=1,-1,0

FlyoutZoomView.zoomfactor

[FlyoutZoomView.|<containerId>_flyout.]zoomfactor=primaryFactor[, [secondaryFactor][, upscale]]

<i>primaryFactor</i>	Specifies the image magnification for the flyout view, relative to the main view. Must be an integer or floating point value 1.0 or larger.
<i>secondaryFactor</i>	An optional secondary factor can be specified which is accessible by clicking or tapping on the main view when the highlight is active. Clicking or tapping a second time reverts to the primary zoom factor. A value of -1 disables the secondary zoom factor.
<i>upscale</i>	<p>Specifies how the component handles small images.</p> <p>If set to 1 the component upscales the main image so that it fits within the main view. Also, it upscales the zoom image so that it completely fills the configured flyout window area.</p> <p>If set to 0 small images are displayed at their original resolution and display centered in the main view area and inside the flyout window. You can configure extra white space around the image with a background or similar CSS property of the <code>s7flyoutzoomview</code> and <code>s7flyoutzoom</code> CSS classes in the main view and flyout window, respectively.</p>

Properties

Optional.

Default

3,-1,1

Example

zoomfactor=2,3,0

Swatches.align

[Swatches.|<containerId>_swatches.]align=left|center|right,top|center|bottom

Specifies internal alignment (anchoring) of the swatches container within the component area. In Swatches, the internal thumbnail container is sized so that only a whole number of swatches is shown. As a result, there is some padding between the internal container and the external component bounds. This command specifies how the internal swatches container is positioned inside the component.

left center right	Sets horizontal swatches alignment.
-------------------	-------------------------------------

top center bottom	Sets vertical swatches alignment.
-----------------------	-----------------------------------

Properties

Optional.

Default

center, center

Example

align=left, top

Swatches.buttonsnapmode

[Swatches. | <containerId>_swatches.]buttonsnapmode=snapin | snapout | overlay

<i>snapin</i>	Causes the buttons to align next to the swatches.
<i>snapout</i>	Causes the buttons to align next to the component border.
<i>overlay</i>	Causes the buttons to render on top of the swatches.

Properties

Optional.

Default

snapout

Example

buttonsnapmode=overlay

Swatches.direction

[Swatches. | <containerId>_swatches.]direction=auto | left | right

auto left right	<p>Specifies the way swatches fill in the view.</p> <p>left sets left-to-right fill order; right reverses the order so that the view is filled in right-to-left, top-to-bottom direction. When auto is set, the component applies right mode when locale is set to "ja", and uses left otherwise.</p>
---------------------	---

Properties

Optional.

Default

auto

Example

direction=right

Swatches.enabledragging

[Swatches.|<containerId>_swatches.]enabledragging=0|1[,overdragvalue]

0 1	Enables or disables the ability for a user to scroll the swatches with a mouse or by using touch gestures
overdragvalue	Functions within the 0–1 range. It is a % value for movement in the wrong direction of the actual speed. If it is set to 1, it moves with the mouse. If it is set to 0, it does not let you move in the wrong direction at all.

Properties

Optional.

Default


1,0.5

Example

enabledragging=0

Swatches.iscommand

[Swatches.|<containerId>_swatches.]iscommand=isCommand

isCommand	<p>The Image Serving command string that is applied to all swatches. If it is specified in the URL, be sure that you HTTP-encode all occurrences of & and = as %26 and %3D, respectively.</p> <p> Note: Image sizing manipulation commands are not supported.</p>
-----------	---

Properties

Optional.

Default

None.

Example

When specified in the viewer URL.

iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000

When specified in the configuration data.

iscommand=op_sharpen=1&op_colorize=0xff0000

Swatches.maxloadradius

[Swatches.|<containerId>_swatches.]maxloadradius=-1|0|preloadnbr

-1 0 preloadnbr	<p>Specifies the component preload behavior. When set to -1 all swatches will be loaded simultaneously when component is initialized or asset changed. When set to 0 only visible swatches are loaded.</p> <p>preloadnbr defines how many invisible rows/columns around the visible area will be preloaded.</p>
-----------------	---

Properties

Optional.

Default

1

Example

maxloadradius=-1

Swatches.pagemode

[Swatches.|<containerId>_swatches.]pagemode=0|1

0 1	<p>When toggled, the scroll buttons automatically cause the swatches to jump a full page length. Extra whitespace is shown on the last page if the swatches do not fit. Also, the last page has the same number of cells as any previous page.</p> <p>The scrollstep is ignored and mouse scrolling settles only on full pages.</p>
-----	---

Properties

Optional.

Default

0

Example

pagemode=1

Swatches.partialswatches

[Swatches.|<containerId>_swatches.]partialswatches=0|1

0 1	Specifies whether the component allows scrolling to stop when any of the swatches are partially visible (scrolling is not aligned). The recommended value is <code>false</code> or 0.
-----	---

Properties

Optional.

Default

0

Example

partialswatches=1

Swatches.scrollstep

[Swatches.|<containerId>_swatches.]scrollstep=*hStep*,*vStep*

<i>hStep</i>	Horizontal step.
<i>vStep</i>	Vertical step.

Specifies the number of swatches to scroll for each click or tap of the corresponding scroll button.

Properties

Optional.

Default

3,3

Example

scrollstep=1,1

JavaScript API reference for Inline Zoom Viewer

The main class of the Inline Zoom Viewer is `FlyoutViewer`. It is declared in the `s7viewers` namespace. This JavaScript API covers constructor, methods, and call backs of this particular class.

In all the following examples, `<instance>` stands for the actual name of the JavaScript viewer object that is instantiated from the `s7viewers.FlyoutViewer` class.

dispose

JavaScript API reference for Inline Zoom Viewer.

`dispose()`

Disposes this viewer instance by releasing all resources used by the viewer logic and deleting all inner objects and components created by the viewer in runtime.

The web page code should also delete the viewer instance variable as well to completely remove the viewer from the web browser memory.

If the web page code has registered event listeners directly on Viewer SDK components used by the viewer—or stored external references to such components—such listeners must be explicitly unregistered by the web page code, and such external component references must be deleted prior to calling `dispose()`.

Do not access the Viewer API any more after `dispose()` is called.

Parameters

None.

Returns

None.

Example

```
<instance>.dispose()
```

FlyoutViewer

JavaScript API reference for Inline Zoom Viewer.

`FlyoutViewer([config])`

Constructor; creates a new Inline Zoom Viewer instance.

Parameters

<i>config</i>	<p><code>{Object}</code> optional JSON configuration object, allows all the viewer settings to pass to the constructor and avoid calling individual setter methods. Contains the following properties:</p> <ul style="list-style-type: none">• <code>containerId</code> – <code>{String}</code> ID of the DOM container (normally a <code>DIV</code>) that the viewer is inserted into. It is not necessary to have the container element created by the time this method is called. However, the container must exist when <code>init()</code> is run. Required.• <code>params</code> – <code>{Object}</code> JSON object with viewer configuration parameters where the property name is either a viewer-specific configuration option or an SDK modifier, and the value of that property is a corresponding settings value. Required.• <code>handlers</code> – <code>{Object}</code> JSON object with viewer event callbacks, where the property name is the name of supported viewer event, and the property value is a JavaScript function reference to an appropriate callback. Optional. <p>See Event callbacks for more information about viewer events.</p>
---------------	--

- `localizedTexts` – {Object} JSON object with localization data. Optional.
See [Localization of user interface elements](#) for more information.
See the *Viewer SDK User Guide* and the example for more information about the object's content.

Returns

None.

Example

```
var inlineZoomViewer = new s7viewers.FlyoutViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
    "config" : "Scene7SharedAssets/Universal_HTML5_Zoom_Inline",
    "contenturl" : "http://s7dl.scene7.com/is/content/",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  },
  "handlers": {
    "initComplete": function() {
      console.log("init complete");
    }
  },
  "localizedTexts": {
    "en": {
      "FlyoutZoomView.TIP_BUBBLE_OVER": "Mouse over to zoom"
    },
    "fr": {
      "FlyoutZoomView.TIP_BUBBLE_OVER": "Passez la souris sur pour zoomer"
    }
  },
  defaultLocale: "en"
});
```

getComponent

JavaScript API reference for Inline Zoom Viewer

`getComponent(componentId)`

Returns a reference to the Viewer SDK component that is used by the viewer. The web page can use this method to extend or customize the behavior of the out-of-box viewer. Call this method only after the `initComplete` viewer callback has run; otherwise, the component may not be created yet by the viewer logic.

Parameters

componentID – {String} an ID of the Viewer SDK component used by the viewer. This viewer supports the following component IDs:

Component ID	Viewer SDK component class name
<code>parameterManager</code>	<code>s7sdk.ParameterManager</code>
<code>container</code>	<code>s7sdk.common.Container</code>

Component ID	Viewer SDK component class name
mediaSet	s7sdk.set.MediaSet
flyout	s7sdk.image.FlyoutZoomView
swatches	s7sdk.set.Swatches

When working with SDK APIs it is important to use correct fully qualified SDK namespace as described in [Viewer SDK](#).

Refer to the Viewer SDK documentation for more information about a particular component.

Returns

{Object} a reference to Viewer SDK component. The method returns null if the componentId is not a supported viewer component or if the component was not yet created by the viewer logic.

Example

```
<instance>.setHandlers({  
  "initComplete":function() {  
    var flyoutZoomView = <instance>.getComponent("flyout");  
  }  
})
```

init

JavaScript API reference for Inline Zoom Viewer.

```
init()
```

Starts the initialization of the viewer so that the viewer code can find it by its ID. By this time the container DOM element must be created.

If the container element is not a part of the web page layout just yet (for example, it may be hidden using `display:none` style assigned to it), the viewer suspends its initialization process until the moment when the web page brings the container element back to the layout. When this occurs, the viewer load automatically resumes.

Only call this method once during the viewer life cycle; subsequent calls are ignored.

Parameters

None.

Returns

{Object} A reference to the viewer instance.

Example

```
<instance>.init()
```

setAsset

JavaScript API reference for Inline Zoom Viewer.

`setAsset (asset)`

<i>asset</i>	<code>{String}</code> new asset id, explicit image set or explicit image set with frame-specific Image Serving modifiers, with optional global Image Serving modifiers appended after <code>?</code> . Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.
--------------	---

Sets the new asset. You can call this parameter at any time, either before or after `init ()`. If it is called after `init ()`, the viewer swaps the asset at runtime.

See also [init](#).

Returns

None.

Example

Single image reference:

```
<instance>.setAsset ( "Scene7SharedAssets/Backpack_B" )
```

Single reference to an image set that is defined in a catalog:

```
<instance>.setAsset ( "Scene7SharedAssets/ImageSet-Views-Sample" )
```

Explicit image set:

```
<instance>.setAsset ( "Scene7SharedAssets/Backpack_B,Scene7SharedAssets/Backpack_C" )
```

Explicit image set with frame-specific Image Serving modifiers:

```
<instance>.setAsset ( "(Scene7SharedAssets/Backpack_B?op_colorize=255%20%20,Scene7SharedAssets/Backpack_B?op_colorize=0x00ff00)" )
```

Sharpening modifier added to all images in the set:

```
<instance>.setAsset ( "Scene7SharedAssets/Backpack_B,Scene7SharedAssets/Backpack_C?op_sharpen=1" )
```

setContainerId

JavaScript API reference for Inline Zoom Viewer.

`setContainerId (containerId)`

Sets the ID of the DOM container (normally a `DIV`) into which the viewer is inserted. It is not necessary to have the container element created by the time this method is called. However, the container must exist when `init ()` is run. It must be called before `init ()`. This method is optional if viewer configuration information is passed with `config` JSON object to the constructor.

<i>containerId</i>	<code>{string}</code> ID of container.
--------------------	--

Returns

None.

Example

```
<instance>.setContainerId ( "s7viewer" );
```

setHandlers

JavaScript API reference for Inline Zoom Viewer.

`setHandlers(handlers)`

Specifies zero or more callback handlers. A call to this method fully overwrites event handlers that were previously assigned for that viewer instance. Must be called before `init()`.

Parameter

<i>handlers</i>	<p>{Object} JSON object with viewer event callbacks, where the property name is the name of the supported viewer event and the property value is a JavaScript function reference to an appropriate callback.</p> <p>See Event callbacks for more information about viewer events.</p>
-----------------	---

Returns

None.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    console.log("init complete");
  }
})
```

setLocalizedTexts

JavaScript API reference for Inline Zoom Viewer.

`setLocalizedTexts(localizationInfo)`

<i>localizationInfo</i>	<p>{Object} JSON object with localization data.</p> <p>See Localization of user interface elements for more information.</p> <p>See the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
-------------------------	---

Sets localization SYMBOL values for one or more locales. This parameter must be called before `init()`.

See also [init](#).

Returns

None.

Example

```
<instance>.setLocalizedTexts({"en":{"FlyoutZoomView.TIP_BUBBLE_OVER":"Mouse over to zoom"},"fr":{"FlyoutZoomView.TIP_BUBBLE_OVER":"Passez la souris sur pour zoomer"},defaultLocale:"en"})
```

setParam

JavaScript API reference for Inline Zoom Viewer.

`setParam(name, value)`

Sets the viewer parameter to a specified value. The parameter is either a viewer-specific configuration option or a software development kit modifier. This parameter is called before `init()`. This method is optional if viewer configuration information is passed with `config` JSON object to the constructor.

See also [init](#).

<i>name</i>	{string} name of parameter.
<i>value</i>	{string} value of parameter. The value cannot be percent-encoded.

Returns

None.

Example

```
<instance>.setParam("style", "customStyle.css")
```

setParams

JavaScript API reference for Inline Zoom Viewer.

`setParams(params)`

<i>params</i>	{string} name=value parameter pairs separated with &.
---------------	---

Sets one or more parameters to a given value. The method argument syntax is identical to a URL query string. That is, it represents name=value pairs separated with &. The same as in a query string, names and values are percent-encoded using UTF8. Before you call `init()`, this parameter must be called. This method is optional if viewer configuration information is passed with `config` JSON object to the constructor.

See also [init](#).

Returns

None.

Example

```
<instance>.setParams("FlyoutZoomView.zoomfactor=2,3&Swatches.iscommand=op_sharpen%3d1")
```

Event callbacks

The viewer supports JavaScript event callbacks that the web page uses to track the viewer initialization process or runtime behavior.

Callback handlers are assigned by passing event names and corresponding handler functions with the `handlers` property to `config` JSON object in the viewer's constructor. Alternatively, it is possible to use `setHandlers()` API method.

Supported viewer events include the following:

- `initComplete` – triggers when viewer initialization is complete and all internal components are created, so that it is possible to use `getComponent()` API. The callback handler does not take any arguments.
- `trackEvent` – triggers each time an event occurs inside the viewer which may be handled by an event tracking system, such as Adobe Analytics. The callback handler takes the following arguments:
 - `objID` {String} not currently used.
 - `compClass` {String} not currently used.
 - `instName` {String} an instance name of the Viewer SDK component that triggered the event.
 - `timeStamp` {Number} event time stamp.
 - `eventInfo` {String} event payload.

See also [FlyoutViewer](#) and [setHandlers](#).

Customizing Inline Zoom Viewer

All visual customization and most behavior customization is done by creating a custom CSS.

The suggested workflow is to take the default CSS file for the appropriate viewer, copy it to a different location, customize it, and specify the location of the customized file in the `style=` command.

Default CSS files can be found at the following location:

```
<s7_viewers_root>/html5/InlineZoomViewer.css
```

Custom CSS file must contain the same class declarations as the default one. If a class declaration is omitted, the viewer does not function properly because it does not provide built-in default styles for user interface elements.

Also it is important to preserve the following CSS declaration from the default viewer CSS as it is needed for inline zoom functionality:

```
.s7flyoutviewer .s7flyoutzoomview .s7flyoutzoom {
  width: 100%;
  height: 100%;
  left: 0;
  top: 0;
  border: none;
  z-index: 999;
}
```

An alternative way to provide custom CSS rules is to use embedded styles directly on the web page or in one of linked external CSS rules.

When you create custom CSS, keep in mind that the viewer assigns `.s7flyoutviewer` class to its container DOM element. If you are using external CSS file passed with `style=` command, use `.s7flyoutviewer` class as parent class in descendant selector for your CSS rules. If you are doing embedded styles on the web page, additionally qualify this selector with an ID of the container DOM element as follows:

```
#<containerId>.s7flyoutviewer
```

Building responsive designed CSS

It is possible to target different devices and embedding sizes in CSS to make your content display differently, depending on a user's device or a particular web page layout. This includes, but is not limited to, different web page layouts, user interface element sizes, and artwork resolution.

The viewer supports two methods for creating responsive designed CSS: CSS markers and standard CSS media queries. You can use these methods independently or together.

CSS markers

To assist in creating responsive designed CSS, the viewer supports CSS markers which special CSS classes dynamically assigned to the top-level viewer container element based on the run-time viewer size and the input type used on the current device.

The first group of CSS Markers includes `.s7size_large`, `.s7size_medium`, and `.s7size_small` classes. They are applied based on the runtime area of the viewer container. That is, if the viewer area is equal to or bigger than the size of a common desktop monitor `.s7size_large` is used; if the area is close in size to a common tablet device `.s7size_medium` is assigned. For areas similar to mobile phone screens `.s7size_small` is set. The primary purpose of these CSS markers is to create different user interface layouts for different screens and viewer sizes.

The second group of CSS Markers includes `.s7mouseinput` and `.s7touchinput`. `.s7touchinput` is set if the current device has touch input capabilities; otherwise, `.s7mouseinput` is used. These markers are intended to create user interface input elements with different screen sizes for different input types, because normally touch input requires larger elements. In cases where the device has both mouse input and touch capabilities, `.s7touchinput` is set and the viewer renders a touch-friendly user interface.

The following sample CSS sets the zoom in button size to 28 x 28 pixels on systems with mouse input, and 56 x 56 pixels on touch devices. In addition, it hides the button completely if the viewer size becomes really small:

```
.s7flyoutviewer.s7mouseinput .s7swatches .s7thumb {
  width: 28px;
  height: 28px;
}
.s7flyoutviewer.s7touchinput .s7swatches .s7thumb {
  width: 56px;
  height: 56px;
}
.s7flyoutviewer.s7size_small .s7swatches {
  visibility: hidden;
}
```

To target devices with a different pixel density, use CSS media queries. The following media query block would contain CSS that is specific to high density screens:

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5)
{
}
```

Using CSS markers is the most flexible way of building responsive designed CSS that lets you to target not only device screen size but actual viewer size, which may be useful for responsive designed web page layouts.

CSS media queries

Device sensing can also be done using pure CSS media queries. Everything enclosed within a given media query block is applied only when it is run on a corresponding device.

When applied to Mobile Viewers, use four CSS media queries, defined in your CSS in the following order:

1. Contains only rules specific for all touch devices.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px)
{
}
```

2. Contains only rules specific for tablets with high resolution screens.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px) and (-webkit-min-device-pixel-ratio:1.5),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px) and (-webkit-min-device-pixel-ratio:1.5)
{
}
```

3. Contains only rules specific for all mobile phones.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in)
{
}
```

4. Contains only rules specific for mobile phones with high resolution screens.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in) and
(-webkit-min-device-pixel-ratio: 1.5),
only screen and (device-width:720px) and (device-height:1280px) and
(-webkit-device-pixel-ratio: 2),
only screen and (device-width:1280px) and (device-height:720px) and
(-webkit-device-pixel-ratio: 2)
{
}
```

Using a media queries approach, you should organize CSS with device sensing as follows:

- First, the desktop-specific section defines all properties that are either desktop-specific or common to all screens.
- And second, the four media queries go in the order defined above and provide CSS rules that are specific for the corresponding device type.

There is no need to duplicate the entire viewer CSS in each media query. Only properties that are specific to given devices are redefined inside a media query.

CSS Sprites

Many viewer user interface elements are styled using bitmap artwork and have more than one distinct visual state. A good example is a button that normally has at least 3 different states: "up", "over", and "down". Each state requires its own bitmap artwork assigned.

With a classic approach to styling, the CSS would have a separate reference to individual image file on the server for each state of the user interface element. The following is a sample CSS for styling a scroll button:

```
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="up"]{
background-image:url(images/v2/ScrollLeftButton_up.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="over"]{
background-image:url(images/v2/ScrollLeftButton_over.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="down"]{
background-image:url(images/v2/ScrollLeftButton_down.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="disabled"]{
```



```
background-image:url(images/v2/ScrollLeftButton_disabled.png);
}
```

The drawback to this approach is that the end user experiences flickering or delayed user interface response when the element is interacted with for the first time. This action occurs because the image artwork for the new element state is not yet downloaded. Also, this approach may have a slight negative impact on performance because of an increase in the number of HTTP calls to the server.

CSS sprites is a different approach where image artwork for all element states is combined into a single PNG file called a "sprite". Such "sprite" has all visual states for the given element positioned one after another. When styling a user interface element with sprites the same sprite image is referenced for all different states in the CSS. Also, the `background-position` property is used for each state to specify which part of the "sprite" image is used. You can structure a "sprite" image in any suitable way. Viewers normally have it vertically stacked. Below is a "sprite"-based example of styling the same scroll button from above:

```
.s7flyoutviewer .s7scrollleftbutton[state] {
    background-image: url(images/v2/ScrollLeftButton_light_sprite.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="up"]{
background-position: -56px -504px;
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="over"]{
background-position: -0px -504px;
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="down"]{
background-position: -56px -448px;
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="disabled"]{
background-position: -0px -448px;
}
```

General styling notes and advice

- When customizing the viewer user interface with CSS the use of the `!important` rule is not supported to style viewer elements. In particular, `!important` rule should not be used to override any default or run-time styling provided by the viewer or Viewer SDK. The reason is that it may affect the behavior of proper components. Instead, you should use CSS selectors with the proper specificity to set CSS properties that are documented in this reference guide.
- All paths to external assets within CSS are resolved against the CSS location, not the viewer HTML page location. Be aware of this rule when you copy the default CSS to a different location. Either copy the default assets as well or update paths within the custom CSS.
- The preferred format for bitmap artwork is PNG.
- Bitmap artwork is assigned to user interface elements using the `background-image` property.
- The width and height properties of a user interface element define its logical size. The size of the bitmap passed to `background-image` does not affect logical size.
- To use the high pixel density of high-resolution screens like Retina, specify bitmap artwork twice as large as the logical user interface element size. Then, apply the `-webkit-background-size:contain` property to scale the background down to the logical user interface element size.
- To remove a button from the user interface, add `display:none` to its CSS class.
- You can use various formats for color value that CSS supports. If you need transparency, use the format `rgba(R,G,B,A)`. Otherwise, you can use the format `#RRGGBB`.

Common User Interface Elements

The following is user interface elements reference documentation that applies to Flyout Viewer:

Main viewer area

The main view area is the area occupied by the flyout view and swatches.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7flyoutviewer
```

CSS property	Description
width	The width of the viewer.
height	The height of the viewer.
background-color	Background color in hexadecimal format.

Example – to set up a flyout viewer with white background (#FFFFFF) and make its size 260 x 500 pixels.

```
.s7flyoutviewer {
  background-color: #FFFFFF;
  width: 260px;
  height: 500px;
}
```

Flyout zoom view

The main view consists of the static image, the zoomed image shown in the flyout view on top of the static image, and the tip message shown on top of static image.

CSS properties of the main view

The appearance of the main view is controlled with the following CSS class selector:

```
.s7flyoutviewer .s7flyoutzoomview
```

CSS property	Description
background-color	The background color of the main view.

Example – to make the main view transparent:

```
.s7flyoutviewer .s7flyoutzoomview {
  background-color: transparent;
}
```

CSS properties of the tip message

The appearance of the tip message is controlled with the following CSS class selector:

```
.s7flyoutviewer .s7flyoutzoomview .s7tip
```

It is possible to configure font style, size appearance and vertical offset through CSS. However, horizontal alignment is managed by the viewer logic. Overriding it through CSS using `left` or `right` properties is not supported.

CSS property	Description
bottom	Offset from the bottom of the main view.
color	Text color.
font-family	Font name.
font-size	Font size.
padding	Padding around the message text.
background-color	Background fill color of message text.
border-radius	Background border radius of message text.
opacity	Background opacity of message text. For Internet Explorer 8, use <code>filter:alpha(opacity=...) </code>

The tip message can be localized. See [Localization of user interface elements](#) for more information.

.

Example – to set up semi-transparent tip message with white Arial 12px font, 50 pixels offset from the bottom of the main view, padding, and a rounded border:

```
.s7flyoutviewer .s7flyoutzoomview .s7tip {
bottom: 50px;
color: #ffffff;
font-family: Arial;
font-size: 12px;
padding-bottom: 10px;
padding-top: 10px;
padding-left: 12px;
padding-right: 12px;
background-color: #000000;
border-radius: 4px;
opacity: 0.5;
filter: alpha(opacity = 50);
}
```

Focus highlight

Input focus highlight displayed around focused viewer UI element is controlled with the CSS class selector.

CSS properties

The appearance is controlled with the following CSS class selector:

```
.s7flyoutviewer *:focus
```

CSS property	Description
outline	Focus highlight style.

Example – to disable the default browser focus highlight for all viewer user interface elements, add the following CSS selector to viewer's style sheet:

```
.s7flyoutviewer *:focus {
  outline: none;
}
```

Swatches

Swatches consist of a row of thumbnail images with optional scroll buttons on the left and right hand side.

Scroll buttons are only visible on the desktop if all thumbnails cannot fit into the width of the container. On mobile devices, or if thumbnails can fit into the container width, scroll buttons are not shown.

CSS properties of the swatches

The appearance of the swatches container is controlled with the following CSS class selector:

```
.s7flyoutviewer .s7swatches
```

CSS property	Description
width	The width of the swatches.
height	The height of the swatches.
bottom	The vertical swatches offset relative to the viewer container.

Example – to set up swatches to 460 x 100 pixels:

```
.s7flyoutviewer .s7swatches {
  width: 460px;
  height: 100px;
}
```

CSS properties of the thumbnail swatch spacing

The spacing between swatch thumbnails is controlled with the CSS class selector:

```
.s7flyoutviewer .s7swatches .s7thumbcell
```

CSS property	Description
margin	The size of the horizontal and vertical margin around each thumbnail. Actual thumbnail spacing equals to the sum of left and right margin set for <code>.s7thumbcell</code> .

Example – to set up spacing to be 10 pixels both vertically and horizontally:

```
.s7flyoutviewer .s7swatches .s7thumbcell {
  margin: 5px;
}
```

CSS properties of the thumbnail swatches

The appearance of individual thumbnail is controlled with the following CSS class selector:

```
.s7flyoutviewer .s7swatches .s7thumb
```

CSS property	Description
width	The width of the thumbnail swatches.
height	The height of the thumbnail swatches.
border	The border of the thumbnail swatches.



Note: Thumbnail supports the *state* attribute selector, which is used to apply different skins to different thumbnail states. In particular, *state="selected"* corresponds to the thumbnail for the image that is currently displayed in the main view, *state="default"* corresponds to the rest of the thumbnails, and *state="over"* is used on mouse hover.

Example – to set up thumbnails that are 56 x 56 pixels, have a light grey default border, and a dark grey selected border:

```
.s7flyoutviewer .s7swatches .s7thumb {
  width: 56px;
  height: 56px;
}
.s7flyoutviewer .s7swatches .s7thumb[state="default"] {
  border: 1px solid #dddddd;
}
.s7flyoutviewer .s7swatches .s7thumb[state="selected"] {
  border: 1px solid #666666;
}
```

CSS properties of the left and right scroll buttons

The appearance of left and right scroll buttons are controlled with the following CSS class selectors:

```
.s7flyoutviewer .s7swatches .s7scrollleftbutton
.s7flyoutviewer .s7swatches .s7scrollrightbutton
```

It is not possible to position scroll buttons using CSS *top*, *left*, *bottom*, and *right* properties. Instead, the viewer logic positions them automatically.

CSS property	Description
width	The width of the scroll button.
height	The height of the scroll button.
background-image	The image that is displayed for a given button state.

CSS property	Description
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the *state* attribute selector, which is used to apply different skins to button states up, down, over, and disabled.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up scroll buttons that are 56 x 56 pixels and have different artwork for each state:

```
.s7flyoutviewer .s7swatches .s7scrollleftbutton {
  background-size: auto;
  width: 56px;
  height: 56px;
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="up"]{
background-image:url(images/v2/ScrollLeftButton_up.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="over"]{
  background-image:url(images/v2/ScrollLeftButton_over.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="down"]{
  background-image:url(images/v2/ScrollLeftButton_down.png);
}
.s7flyoutviewer .s7swatches .s7scrollleftbutton[state="disabled"]{
  background-image:url(images/v2/ScrollLeftButton_disabled.png);
}
.s7flyoutviewer .s7swatches .s7scrollrightbutton {
  background-size: auto;
  width: 56px;
  height: 56px;
}
.s7flyoutviewer .s7swatches .s7scrollrightbutton[state="up"]{
background-image:url(images/v2/ScrollRightButton_up.png);
}
.s7flyoutviewer .s7swatches .s7scrollrightbutton[state="over"]{
  background-image:url(images/v2/ScrollRightButton_over.png);
}
.s7flyoutviewer .s7swatches .s7scrollrightbutton[state="down"]{
  background-image:url(images/v2/ScrollRightButton_down.png);
}
.s7flyoutviewer .s7swatches .s7scrollrightbutton[state="disabled"]{
  background-image:url(images/v2/ScrollRightButton_disabled.png);
}
```

Tooltips

On desktop systems some user interface elements like buttons have tooltips that are displayed on mouse hover.

CSS properties of the main viewer area

The appearance of tooltips is controlled with the following CSS class selector:

```
.s7tooltip
```

CSS property	Description
border-radius	Background border radius.
border-color	Background border color.
background-color	Background color.
color	Text color.
font-family	Text font name.
font-size	Text font size.



Note: In case tooltip styles are customized from within the embedding web page, all properties have to contain !IMPORTANT rule. This is not necessary if tooltips are customized in the viewer's CSS file.

Example – to set up tooltips that have a grey border with 3px corner radius, black background and white text written with Arial, 11 pixels size:

```
.s7tooltip {
border-radius: 3px 3px 3px 3px;
border-color: #999999;
background-color: #000000;
color: #FFFFFF;
font-family: Arial, Helvetica, sans-serif;
font-size: 11px;
}
```

Support for Adobe Analytics tracking

The Flyout Viewer supports Adobe Analytics tracking out of the box.

Out-of-the-box tracking

The Inline Zoom Viewer supports Adobe Analytics tracking out-of-the-box. To enable tracking, pass the proper company preset name as config2 parameter.

The viewer also sends a single tracking HTTP request to the configured Image Server with the viewer type and version information.

Custom tracking

To integrate with third-party analytics systems it is necessary to listen to the trackEvent viewer callback and process the eventInfo argument of the callback function as necessary. The following code is an example of such handler function:

```
var inlineZoomViewer = new s7viewers.FlyoutViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
    "config": "Scene7SharedAssets/Universal_HTML5_Zoom_Inline",
    "contenturl": "http://s7dl.scene7.com/is/content/",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  },
  "handlers": {
```

```

"trackEvent":function(objID, compClass, instName, timeStamp, eventInfo) {
  //identify event type
  var eventType = eventInfo.split(",")[0];
  switch (eventType) {
    case "LOAD":
      //custom event processing code
      break;
      //additional cases for other events
  }
}
});

```

The viewer tracks the following SDK user events:

SDK user event	Sent when...
LOAD	the viewer is loaded first.
SWAP	an asset is swapped in the viewer using <code>setAsset()</code> API.
ZOOM	the flyout is activated or the zoom level is changed.
PAN	an image is panned.
SWATCH	an image is changed by clicking or tapping on a swatch.

Localization of user interface elements

Certain content that the Flyout Viewer displays is subject to localization. This content includes user interface element tool tips and information messages that are displayed by the flyout zoom view on load.

Every textual content in the viewer that can be localized is represented by the special Viewer SDK identifier called SYMBOL. Any SYMBOL has a default associated text value for an English locale ("en") supplied with the out-of-the-box viewer, and also may have user-defined values set for as many locales as needed.

When the viewer starts, it checks the current locale to see if there is a user-defined value for each supported SYMBOL for such locale. If there is, it uses the user-defined value; otherwise, it falls back to the out-of-the-box default text.

User-defined localization data can be passed to the viewer as a localization JSON object. Such object contains the list of supported locales, SYMBOL text values for each locale, and the default locale.

An example of such a localization object is the following:

```

{
  "en":{
    "FlyoutZoomView.TIP_BUBBLE_OVER":"Mouse over to zoom",
    "FlyoutZoomView.TIP_BUBBLE_TAP":"Tap and hold to zoom"
  },
  "fr":{
    "FlyoutZoomView.TIP_BUBBLE_OVER":"Passez la souris sur pour zoomer",
    "FlyoutZoomView.TIP_BUBBLE_TAP":"Appuyez et maintenez pour agrandir"
  },
  defaultLocale:"en"
}

```


In the above example, the localization object defines two locales ("en" and "fr") and provides localization for two user interface elements in each locale.

The web page code should pass such localization object to the viewer constructor, as a value of the `localizedTexts` field of the configuration object. An alternative option is to pass the localization object by calling the `setLocalizedTexts(localizationInfo)` method.

The following SYMBOLs are supported:

SYMBOL	Description
<code>FlyoutZoomView.TIP_BUBBLE_OVER</code>	Information message for desktop systems.
<code>FlyoutZoomView.TIP_BUBBLE_TAP</code>	Information message for touch devices.
<code>ScrollLeftButton.TOOLTIP</code>	Tooltip for scroll left button.
<code>ScrollRightButton.TOOLTIP</code>	Tooltip for scroll right button.
<code>ScrollUpButton.TOOLTIP</code>	Tooltip for scroll up button.
<code>ScrollDownButton.TOOLTIP</code>	Tooltip for scroll down button.

Viewer SDK namespace

The viewer is built of many Viewer SDK components. In most cases, the web page does not need to interact with SDK components API directly; all common needs are covered in the viewer API itself.

However, some advanced use cases require that the web page obtain a reference to an inner SDK component using the `getComponent()` viewer API and then use all the flexibility of the APIs of SDK itself.

The namespace that is used to load and initialize SDK components by the viewer depends on the environment in which the viewer is operating. If the viewer is running in AEM (Adobe Experience Manager), the viewer loads SDK components into `s7viewers.s7sdk` namespace. Likewise, the viewer served from Scene7 Publishing System loads the SDK into `s7classic.s7sdk`.

In either case, the namespace used by the SDK inside the viewer has either `s7viewers` or `s7classic` as the prefix. And, it is different from the plain `s7sdk` namespace used in the SDK User Guide or SDK API documentation.

For that reason, it is important to use a fully qualified SDK namespace when you write custom application code that communicates with internal viewer components.

For example, if you plan to listen to `StatusEvent.NOTF_VIEW_READY` event and the viewer is served from AEM, the fully qualified event type is `s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY`, and the event listener code looks similar to the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var flyout = <instance>.getComponent("flyout");
    flyout.addEventListener(s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
```

```

        console.log("view ready");
    }, false);
}
});

```

The same code for viewer served from Scene7 SPS will look like this:

```

<instance>.setHandlers({
  "initComplete":function() {
    var flyout = <instance>.getComponent("flyout");
    flyout.addEventListener(s7classic.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});

```

Mixed Media

Mixed Media Viewer is a media viewer. It supports media sets that contain images, swatch sets, spin sets, videos, and Adaptive Video Sets.

A thumbnail at the bottom of the viewer represents each media set element, along with its asset type indicator. When a swatch set element is selected, a secondary row of swatches appears to allow the selection of color variation within the swatch set. Images and swatch set elements support zooming in continuous or inline mode; spin sets support both zooming and spinning. Videos and Adaptive Video Sets support all basic playback controls as long as any optional closed captions are displayed on top of the video content. A user can switch to full screen any time by clicking the full screen button. The viewer has optional close button. It is designed to work on desktops and mobile devices.

The Mixed Media Viewer uses HTML5 streaming video playback in HLS format in its default configuration whenever the underlying system supports that. On systems that do not support HTML5 streaming the viewer falls back to HTML5 progressive video delivery. Finally, the viewer still supports Flash-based streaming video playback if requested using the `VideoPlayer.playback` configuration parameter.



Note: Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.

Viewer type 505.

See [System requirements](#).

Demo URL

https://s7d9.scene7.com/s7viewers/html5/MixedMediaViewer.html?asset=Scene7SharedAssets/Mixed_Media_Set_Sample

Using Mixed Media Viewer

Mixed Media Viewer represents a main JavaScript file and a set of helper files (a single JavaScript include with all Viewer SDK components used by this particular viewer, assets, CSS) downloaded by the viewer in runtime.

You can use the Mixed Media Viewer in pop-up mode using production-ready HTML page provided with IS-Viewers. Or, you can use the viewer in embedded mode, where it is integrated into a target web page using the documented API.

The task of configuring and skinning the viewer is similar to other viewers. All skinning is achieved by way of custom CSS.

See [Command reference common to all viewers – Configuration attributes](#) and [Command reference common to all Viewers – URL](#)

Interacting with Mixed Media Viewer

Mixed Media Viewer supports single-touch and multi-touch gestures that are common in other mobile applications. When the viewer cannot process a user's swipe gesture it forwards the event to the web browser to perform a native page scroll. This functionality lets a user navigate through the page even if the viewer occupies most of the device screen area.

Gesture	Description
Single tap	Activates the flyout view or changes between the primary and secondary zoom level.
Double tap	When in <code>continuous</code> zoom mode, zooms in one level until maximum magnification is reached, the next double tap gesture resets to initial state.
Touch and hold	When in <code>inline</code> zoom mode, activates zoomed image.
Pinch	When in <code>continuous</code> zoom mode, zooms image in or out.
Horizontal swipe or flick	<p>When the current asset is a spin set, and the image is in a reset state, it spins through the spin set horizontally.</p> <p>When the current asset is a spin set or an image and the image is zoomed in, it moves the image horizontally. If the image is moved to the view edge and swipe is still done in that direction, the gesture performs a native page scroll.</p> <p>Scrolls through the list of swatches in the swatch bar.</p>
Vertical swipe or flick	<p>If the image is in a reset state, it changes the vertical view angle in case a multi-dimensional spin set is used. In a one-dimensional spin set, or when a multi-dimensional spin set is on the last or first axis, so that vertical swipe does not result in vertical view angle change, the gesture performs native page scroll.</p> <p>When current asset is a spin set or an image and the image is zoomed in, moves the image vertically. If image is moved to the view edge and swipe is still done in that direction, the gesture performs native page scroll.</p> <p>If the gesture is done within the swatches area, it performs a native page scroll.</p>

The viewer also supports both touch input and mouse input on Windows devices with touch screen and mouse. This support, however, is limited to Chrome, Internet Explorer 11, and Edge web browsers only.

This viewer is fully keyboard accessible.

See [Keyboard accessibility and navigation](#).

Embedding Mixed Media Viewer

Different web pages have different needs for viewer behavior. Sometimes a web page provides a link that, when clicked, opens the viewer in a separate browser window. In other cases, it is necessary to embed the viewer right in the hosting page. In the latter case, the web page may have a static page layout, or use responsive design that displays differently on different devices or

for different browser window sizes. To accommodate these needs, the viewer supports three primary operation modes: pop-up, fixed size embedding, and responsive design embedding.

About pop-up mode

In pop-up mode, the viewer is opened in a separate web browser window or tab. It takes the entire browser window area and adjusts in case the browser is resized, or a mobile device's orientation is changed.

Pop-up mode is the most common for mobile devices. The web page loads the viewer using `window.open()` JavaScript call, properly configured A HTML element, or any other suitable method.

It is recommended that you use an out-of-the-box HTML page for pop-up operation mode. In this case, it is called `MixedMediaViewer.html` and is located within the `html5/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/MixedMediaViewer.html
```

You can achieve visual customization by applying custom CSS.

The following is an example of HTML code that opens the viewer in a new window:

```
<a  
href="http://s7dl.scene7.com/s7viewers/html5/MixedMediaViewer.html?asset=Scene7SharedAssets/Mixed_Media_Set_Sample"  
target="_blank">Open popup viewer</a>
```

About fixed size and responsive design embedding

In the embedded mode, the viewer is added to the existing web page, which may already have some customer content not related to the viewer. The viewer normally occupies only a part of a web page's real estate.

The primary use cases are web pages oriented for desktops or tablet devices, and also responsive design pages that adjust layout automatically depending on the device type.

Fixed size embedding is used when the viewer does not change its size after initial load. This is the best choice for web pages that have a static layout.

Responsive design embedding assumes that the viewer may need to resize at runtime in response to the size change of its container `DIV`. The most common use case is adding a viewer to a web page that uses a flexible page layout.

In responsive design embedding mode, the viewer behaves differently depending on the way web page sizes its container `DIV`. If the web page sets only the width of the container `DIV`, leaving its height unrestricted, the viewer automatically chooses its height according to the aspect ratio of the asset that is used. This functionality ensures that the asset fits perfectly into the view without any padding on the sides. This use case is the most common for web pages using responsive design layout frameworks like Bootstrap, Foundation, and so on.

Otherwise, if the web page sets both the width and the height for the viewer's container `DIV`, the viewer fills just that area and follows the size that the web page layout provides. A good example is embedding the viewer into a modal overlay, where the overlay is sized according to web browser window size.

Fixed Size Embedding

You add the viewer to a web page by doing the following:

1. Adding the viewer JavaScript file to your web page.
2. Defining the container `DIV`.
3. Setting the viewer size.
4. Creating and initializing the viewer.

1. Adding the viewer JavaScript file to your web page.

Creating a viewer requires that you add a script tag in the HTML head. Before you can use the viewer API, be sure that you include `MixedMediaViewer.js`. The `MixedMediaViewer.js` file is located under the `html5/js/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/js/MixedMediaViewer.js
```

You can use a relative path if the viewer is deployed on one of the Adobe Scene7 servers and it is served from the same domain. Otherwise, you specify a full path to one of Adobe Scene7 servers that have the IS-Viewers installed.

The relative path looks like the following:

```
<script language="javascript" type="text/javascript"
src="/s7viewers/html5/js/MixedMediaViewer.js"></script>
```



Note: You should only reference the main viewer JavaScript *include* file on your page. You should not reference any additional JavaScript files in the web page code which might be downloaded by the viewer's logic in runtime. In particular, do not directly reference `HTML5 SDK Utils.js` library loaded by the viewer from `/s7viewers` context path (so-called consolidated SDK *include*). The reason is that the location of `Utils.js` or similar runtime viewer libraries is fully managed by the viewer's logic and the location changes between viewer releases. Adobe does not keep older versions of secondary viewer *includes* on the server.

As a result, putting a direct reference to any secondary JavaScript *include* used by the viewer on the page breaks the viewer functionality in the future when a new product version is deployed.

2. Defining the container DIV.

Add an empty DIV element to the page where you want the viewer to appear. The DIV element must have its ID defined because this ID is passed later to the viewer API. The DIV has its size specified through CSS.

The placeholder DIV is a positioned element, meaning that the `position` CSS property is set to `relative` or `absolute`.

Ensure that the full screen feature functions properly in Internet Explorer. Check to make sure that there are no other elements in the DOM that have a higher stacking order than your placeholder DIV.

The following is an example of a defined placeholder DIV element:

```
<div id="s7viewer" style="position:relative"></div>
```

3. Setting the viewer size

This viewer displays thumbnails when working with multi-item sets. On desktop systems, thumbnails are placed below the main view. At the same time, the viewer allows the swapping of the main asset during runtime using `setAsset()` API. As a developer, you have control over how the viewer manages the thumbnails area at the bottom when the new asset has only one item. It is possible to keep the outer viewer size intact and let the main view increase its height and occupy the thumbnails area. Or, you can keep the main view size static and collapse the outer viewer area, letting web page content to move up, and then use the free page real estate that is left from the thumbnails.

To keep the outer viewer bounds intact define the size for `.s7mixedmediaviewer` top-level CSS class in absolute units. Sizing in CSS can be put right on the HTML page, or in a custom viewer CSS file, which is later assigned to a viewer preset record in Scene7 Publishing System, or passed explicitly using `style` command.

See [Customizing Mixed Media Viewer](#) for more information about styling the viewer with CSS.

The following is an example of defining the static outer viewer size in an HTML page:

```
#s7viewer.s7mixedmediaviewer {
  width: 640px;
  height: 480px;
}
```

You can see the behavior with a fixed outer viewer area on the following sample page. Notice that when you switch between sets, the outer viewer size does not change:

https://marketing.adobe.com/resources/help/en_US/s7/viewers_ref/samples/MixedMediaViewer-fixed-outer-area.html

To make the main view dimensions static, define the viewer size in absolute units for the inner Container SDK component using `.s7mixedmediaviewer .s7container` CSS selector, or by using `stagesize` modifier.

The following is an example of defining the viewer size for the inner Container SDK component so that the main view area does not change its size when switching the asset:

```
#s7viewer.s7mixedmediaviewer .s7container {
  width: 640px;
  height: 480px;
}
```

The following sample page shows viewer behavior with a fixed main view size. Notice that when you switch between sets, the main view remains static and the web page content moves vertically:

https://marketing.adobe.com/resources/help/en_US/s7/viewers_ref/samples/MixedMediaViewer-fixed-main-view.html

You can set the `stagesize` modifier either in the viewer preset record in Scene7 Publishing System, or pass it explicitly with the viewer initialization code with `params` collection, or as an API call as described in the Command Reference section of this Help, as in the following:

```
mixedMediaViewer.setParam("stagesize", "640,480");
```

A CSS-based approach is recommended and is used in this example.

4. Creating and initializing the viewer.

When you have completed the steps above, you create an instance of `s7viewers.MixedMediaViewer` class, pass all configuration information to its constructor and call `init()` method on a viewer instance. Configuration information is passed to the constructor as a JSON object. At minimum, this object should have the `containerId` field which holds the name of viewer container ID and nested `params` JSON object with configuration parameters that the viewer supports. In this case, the `params` object must have at least the Image Serving URL passed as `serverUrl` property, the video server URL passed as `videoserverurl` property and initial asset as `asset` parameter. JSON-based initialization API lets you create and start the viewer with single line of code.

It is important to have the viewer container added to the DOM so that the viewer code can find the container element by its ID. Some browsers delay building DOM until the end of the web page. For maximum compatibility, call the `init()` method just before the closing `BODY` tag, or on the `body onload()` event.

At the same time, the container element should not necessarily be part of the web page layout just yet. For example, it may be hidden using `display:none` style assigned to it. In this case, the viewer delays its initialization process until the moment when the web page brings the container element back to the layout. When this occurs, the viewer load automatically resumes.

The following is an example of creating a viewer instance, passing the minimum necessary configuration options to the constructor, and calling the `init()` method. The example assumes `mixedMediaViewer` is the viewer instance; `s7viewer` is the name of placeholder `DIV`; `http://s7d1.scene7.com/is/image/` is the Image Serving URL;

`http://s7dl.scene7.com/is/content/` is the video server URL; and
`Scene7SharedAssets/Mixed_Media_Set_Sample` is the asset:

```
<script type="text/javascript">
var mixedMediaViewer = new s7viewers.MixedMediaViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Scene7SharedAssets/Mixed_Media_Set_Sample",
    "serverurl":"http://s7dl.scene7.com/is/image/",
    "videoserverurl":"http://s7dl.scene7.com/is/content/"
  }
}).init();
</script>
<script type="text/javascript">
mixedMediaViewer.init();
</script>
```

The following code is a complete example of a trivial web page that embeds the Mixed Media Viewer with a fixed size:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/MixedMediaViewer.js"></script>
<style type="text/css">
#s7viewer.s7mixedmediaviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var mixedMediaViewer = new s7viewers.MixedMediaViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Scene7SharedAssets/Mixed_Media_Set_Sample",
    "serverurl":"http://s7dl.scene7.com/is/image/",
    "videoserverurl":"http://s7dl.scene7.com/is/content/"
  }
}).init();
</script>
</body>
</html>
```

Responsive embedding with unrestricted height

With responsive design embedding, the web page normally has some kind of flexible layout in place that dictates the runtime size of the viewer's container `DIV`. For the following example, assume that the web page allows the viewer's container `DIV` to take 40% of the web browser window size, leaving its height unrestricted. The web page HTML code would look like the following:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
.holder {
  width: 40%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>
```

Adding the viewer to such a page is similar to the steps for fixed size embedding. The only difference is that you do not need to explicitly define the viewer size.

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Creating and initializing the viewer.

All the steps above are the same as with the fixed size embedding. Add the container DIV to the existing "holder" DIV. The following code is a complete example. Notice how viewer size changes when the browser is resized, and how the viewer aspect ratio matches the asset.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/MixedMediaViewer.js"></script>
<style type="text/css">
.holder {
width: 40%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var mixedMediaViewer = new s7viewers.MixedMediaViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/Mixed_Media_Set_Sample",
    "serverurl": "http://s7dl.scene7.com/is/image/",
    "videoseverurl": "http://s7dl.scene7.com/is/content/"
  }
}).init();
</script>
</body>
</html>
```

The following examples page illustrates more real-life uses of responsive design embedding with unrestricted height:

https://marketing.adobe.com/resources/help/en_US/s7/vlist/vlist.html

Flexible size embedding with width and height defined

In case of flexible-size embedding with width and height defined, the web page styling is different. It provides both sizes to the "holder" DIV and centers it in the browser window. Also, the web page sets the size of the HTML and BODY element to 100 percent.

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
width: 100%;
height: 100%;
}
.holder {
position: absolute;
left: 20%;
top: 20%;
width: 60%;
height: 60%;
}
```



```

}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>

```

The rest of the embedding steps are identical to the steps used for responsive design embedding with unrestricted height. The resulting example is the following:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/MixedMediaViewer.js"></script>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var mixedMediaViewer = new s7viewers.MixedMediaViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Scene7SharedAssets/Mixed_Media_Set_Sample",
    "serverurl":"http://s7dl.scene7.com/is/image/",
    "videoseverurl":"http://s7dl.scene7.com/is/content/"
  }
}).init();
</script>
</body>
</html>

```

Embedding using Setter-based API

Instead of using JSON-based initialization, it is possible to use setter-based API and no-args constructor. Using this API constructor does not take any parameters and configuration parameters are specified using `setContainerId()`, `setParam()`, and `setAsset()` API methods, with separate JavaScript calls.

The following example illustrates using fixed size embedding with setter-based API:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/MixedMediaViewer.js"></script>
<style type="text/css">
#s7viewer.s7mixedmediaviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>

```

```
<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var mixedMediaViewer = new s7viewers.MixedMediaViewer();
mixedMediaViewer.setContainerId("s7viewer");
mixedMediaViewer.setParam("serverurl", "http://s7dl.scene7.com/is/image/");
mixedMediaViewer.setAsset("Scene7SharedAssets/Mixed_Media_Set_Sample");
mixedMediaViewer.init();
</script>
</body>
</html>
```

Command reference – Configuration attributes

Configuration attributes documentation for Mixed Media Viewer.

Any configuration command can be set in URL or using `setParam()`, or `setParams()`, or both, API methods. Any config attribute can be also specified in server-side configuration record.

Some configuration commands may be prefixed with the class name or instance name of the corresponding Viewer SDK component. An instance name of the component is dynamic and depends on the ID of the viewer container DOM element passed to `setContainerId()` API method. Documentation includes an optional prefix for such commands. For example, `zoomstep` command is documented as follows:

```
[ZoomView.|<containerId>_zoomView].zoomstep
```

which means that you can use this command as:

- `zoomstep` (short syntax)
- `ZoomView.zoomstep` (qualified with component class name)
- `cont_zoomView.zoomstep` (qualified with component ID, assuming `cont` is the ID of the container element)

See also [Command reference common to all viewers – Configuration attributes](#)

closebutton

```
closebutton=0|1
```

0-1	Set to 1 to enable close button display, or set to 0 to hide close button.
-----	--

Properties

Optional.

Default

0

Example

```
closebutton=1
```

ControlBar.transition

```
[ControlBar.|<containerId>_controls.]transition=none|fade[,delaytohide[,duration]]
```

<code>none fade</code>	<p>Specifies the effect type that is used to show or hide the control bar and its content.</p> <p>Use <code>none</code> for instant show and hide. Use <code>fade</code> to provide a gradual fade in and fade out effect.</p> <p>Fade is not supported on Internet Explorer 8.</p>
<code>delaytohide</code>	<p>Specifies the time in seconds between the last mouse/touch event that the control bar registers and the time control bar hides.</p> <p>If set to <code>-1</code> the component never triggers its auto-hide effect and always stays visible on the screen.</p>
<code>duration</code>	Sets the duration of the fade in and fade out animation, in seconds.

Properties

Optional.

Default

`fade, 2, 0.5`

Example

`transition=none`

FlyoutZoomView.fmt

Specifies the image format that the component uses for loading images from Image Server.

`[FlyoutZoomView. | <containerId>_flyout.]fmt=jpg | jpeg | png | png-alpha | gif | gif-alpha`

<code>jpg jpeg png png-alpha gif gif-alpha</code>	<p>If the specified format ends with <code>-alpha</code>, the component renders images as transparent content. For all other image formats, the component treats images as opaque.</p> <p>Note that the component has a white background by default. Therefore, to make it completely transparent, set the <code>background-color</code> CSS property to <code>transparent</code>.</p>
---	--

Properties

Optional.

Default

`jpeg`

Example

`fmt=png-alpha`

FlyoutZoomView.imagereload

Configures how the component fetches new images for the main and flyout view during resize.

[FlyoutZoomView. | <containerId>_flyout.]imagereload=0|1[,breakpoint,width[:width]]

0 1	<p>When set to 0, the component does not load new images during resize, and image resolution in the flyout view does not change.</p> <p>When set to 1 lets you specify one or more width breakpoints for the image loaded into the main view.</p>
breakpoint,width[:width]	Width breakpoints for the image loaded into main view. The component will always use best fit size for the initial load. After resize it will ensure that the image in main view is always downloaded with the width equal to closest bigger breakpoint, and downscaled on the client.

Properties

Optional.

Default


0

Example

imagereload=1,breakpoint,200;400;800;1600

FlyoutZoomView.preloadtiles

[FlyoutZoomView. | <containerId>_flyout.]preloadtiles=0|1

0 1	<p>Set to 1 to enable preloading of the zoomed image.</p> <p>Set to 0 to load the zoom image incrementally, as needed.</p> <p> Note: Be aware that if you enable this option it can result in substantially higher bandwidth usage because the zoomed image must be loaded in its entirety—even if no zoom action is taken by the user.</p>
-----	---

Properties

Optional.

Default

0

Example

preloadtiles=1

FlyoutZoomView.tip

```
[FlyoutZoomView. | <containerId>_flyout. ]tip=duration[ ,count][ ,fade]
```

<i>duration</i>	Specifies the number of seconds the tip text is displayed before it hides. When set to -1, the message is always displayed, even if the user activates the flyout.
<i>count</i>	Specifies the number of times the text is displayed when viewing new images in the set. A value of -1 means that the text is always displayed when viewing any image in the set.
<i>fade</i>	Specifies the duration of a fade animation that occurs when the text appears or disappears. A value of 0 indicates no fade transition.

Properties

Optional.

Default

3,1,0.3

Example

tip=1,-1,0

SpinView.doubleclick

```
[SpinView. | <containerId>_spinView. ]doubleclick=none | zoom | reset | zoomReset
```

none zoom reset zoomReset	Configures the mapping of double-click/tap to spin actions. Setting to <i>none</i> disables double-click/tap spin. If set to <i>zoom</i> clicking the image spins in one spin step; CTRL+Click spins out one spin step. Setting to <i>reset</i> causes a single click on the image to reset the spin to the initial spin level. For <i>zoomReset</i> , reset is applied if the current spin factor is at or beyond the specified limit, otherwise spin is applied.
---------------------------------	--

Properties

Optional.

Default

reset on desktop computers; zoomReset on touch devices.

Example

doubleclick=zoom

SpinView.enableHD

```
[SpinView. | <containerId>_spinView. ]enableHD=always | never | limit[ ,number]
```

<code>always never limit</code>	Enable, limit or disable optimization for devices where <code>devicePixelRatio</code> is greater than 1, that is devices with a high-density display like iPhone4 and similar devices. If active then the component limits the size of the IS image request as if the device only had a pixel ratio of 1 and therefore reducing the bandwidth.
<code>number</code>	If using the <code>limit</code> setting, the component enables high pixel density only up to the specified limit.

Properties

Optional.

Default

`limit,1500`

Example

`enableHD=always`

SpinView.fmt

`[SpinView.|<containerId>_spinView.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha`

<code>jpg jpeg png png-alpha gif gif-alpha</code>	<p>Specifies the image format that the component uses for loading images from Image Server. If the specified format ends with <code>-alpha</code>, the component renders images as transparent content. For all other image formats the component treats images as opaque.</p> <p>The component has a white background by default. Therefore, to make it transparent, set the <code>background-color</code> CSS property to transparent.</p>
---	--

Properties

Optional.

Default

`jpeg`

Example

`fmt=png-alpha`

SpinView.iconeffect

`[SpinView.|<containerId>_spinView.]iconeffect=0|1[,count][,fade][,autoHide]`

<code>0 1</code>	Enables the <code>iconeffect</code> to display on the top of the image when the image is in a reset state and it is suggestive of an available action to interact with the image.
------------------	---

<i>count</i>	Specifies the maximum number of times the <code>iconeffect</code> appears and reappears. A value of -1 indicates that the icon always reappears indefinitely.
<i>fade</i>	Specifies the duration of the show or hide animation, in seconds.
<i>autoHide</i>	Sets the number of seconds that the <code>iconeffect</code> stays fully visible before it auto hides. That is, the time after the fade-in animation is completed but before the fade out animation starts. A setting of 0 disables the auto-hide behavior.

Properties

Optional.

Default


1,1,0.3,3

Example

`iconeffect=0`

SpinView.iscommand

`[SpinView.<containerId>_spinView.iscommand=isCommand`

<i>iscommand</i>	<p>The Image Serving command string that is applied to spin image. If specified in the URL all occurrences of & and = must be HTTP-encoded as %26 and %3D, respectively.</p> <p> Note: Image sizing manipulation commands are not supported.</p>
------------------	--

Properties

Optional.

Default

None.

Example

When specified in the viewer URL:

`iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000`

When specified in the config data:

`iscommand=op_sharpen=1&op_colorize=0xff0000`

SpinView.lockdirection

[SpinView.<containerId>_spinView.lockdirection=0|1

0 1	<p>Specifies whether to allow a change in the spin direction in case of 2D spin set.</p> <p>When set to 1, the component identifies the primary drag or swipe direction (horizontal versus vertical) at the start of the gesture. After that, it maintains that direction until the gesture ends. For example, if the user starts a horizontal spin and then decides to continue their drag gesture in a vertical direction, the component does not perform a vertical spin; instead, it considers only the horizontal movement of the mouse or swipe.</p> <p>A value of 0 lets a user change the spin direction at any time during the gesture progress. The setting has no affect if the spin set is 1D.</p>
-----	--

Properties

Optional.

Default

0

Example

lockdirection=1

SpinView.maxloadradius

Represents the maximum number of frames to preload in each direction when the SpinView is idle.

[SpinView.<containerId>_spinView.maxloadradius=value[,highRes]

value	<p>A value of -1 preloads all frames in the set. The preloaded frames are always seen at the original resolution that the SpinView was initially loaded.</p>
highRes	<p>Controls the quality of preloaded frames.</p> <p>When set to 1 the frames load in high-quality, matching the size of the component.</p> <p>When set to 0 only the low-resolution preview tile is loaded.</p> <p>Preloading in high resolution improves end user experience, especially when auto-spin is enabled. At the same time, it results in slower start time and higher network consumption, so it should be used with care. When high-resolution preload is used, the preloaded frames are always in the original resolution at which the component was initially loaded.</p>

Properties

Optional.

Default

6,0

Example

maxloadradius=12,1

SpinView.sensitivity

[SpinView.|<containerId>_spinView.]sensitivity=xSensitivity[, ySensitivity]

<code>xSensitivity[, ySensitivity]</code>	<p>Controls the sensitivity of the horizontal and vertical spin performed with a mouse drag or swipe.</p> <p><code>xSensitivity</code> sets how many full horizontal product rotations are made if the user drags their mouse horizontally from one side of the view to the other. For example, three means that the user sees three complete spins for one full drag gesture.</p> <p>Likewise, <code>ySensitivity</code> controls the sensitivity of the vertical spin. A value of 1 means that one full vertical drag or swipe changes the view angle from the top-most spin plane to the bottom-most (or vice versa).</p> <p>Setting a negative value for <code>ySensitivity</code> reverses the direction of the vertical spin.</p>
---	---

Properties

Optional.

Default

5,2

Example

sensitivity=4,-2

SpinView.singleclick

[SpinView.|<containerId>_spinView.]singleclick=none|zoom|reset|zoomReset

<code>none zoom reset zoomReset</code>	<p>Configures the mapping of single-click/tap to zoom actions. Setting to <code>none</code> disables single-click/tap zoom. If set to <code>zoom</code> clicking the image zooms in one zoom step; CTRL+Click zooms out one zoom step. Setting to <code>reset</code> causes a single click on the image to reset the zoom to the initial spin level. For <code>zoomReset</code>, reset is applied if the current zoom factor is at or beyond the specified limit, otherwise zoom is applied.</p>
--	--

Properties

Optional.

Default

zoomReset on desktop computers; none on touch devices.

Example

singleclick=zoom

SpinView.transition

[SpinView.|<containerId>_spinView.]transition=*time*[,*easing*]

<i>time</i>	Specifies the time in seconds that the animation for a single zoom step action takes.
<i>easing</i>	<p>Creates an illusion of acceleration or deceleration which makes the transition appear more natural. You can set easing to one of the following:</p> <ul style="list-style-type: none">• 0 (auto)• 1 (linear)• 2 (quadratic)• 3 (cubic)• 4 (quartic)• 5 (quintic) <p>Auto mode always uses linear transition when elastic zoom is disabled (default). Otherwise, it fits one of the other easing functions based on the transition time. That is, the shorter the transition time the higher the easing function is used to accelerate the acceleration or deceleration effect.</p>

Properties

Optional.

Default

0.5,0

Example

transition=2,2

SpinView.zoomstep

[SpinView.|<containerId>_spinView.]zoomstep=*step*[,*limit*]

<i>step</i>	Configures the number zoom in and zoom out actions that are required to increase or decrease the resolution by a factor of two. The resolution change for each zoom action is 2 ¹ per step. Set to 0 to zoom to full resolution with a single zoom action.
-------------	---

<i>limit</i>	Specifies the maximum zoom resolution, relative to the full resolution image. The default is 1.0, which does not allow zooming beyond full resolution.
--------------	--

Properties

Optional.

Default

1,1

Example

zoomstep=2,3

Swatches.align

[Swatches.|<containerId>_swatches.]align=left|center|right,top|center|bottom

Specifies the internal alignment (anchoring) of the swatches container within the component area. In Swatches, the internal thumbnail container is sized so that only a whole number of swatches is shown. As a result, there is some padding between internal container and external component bounds. This command specifies how the internal swatches container is positioned inside component.

left center right	Sets the alignment of the horizontal swatches.
top center bottom	Sets the alignment of the vertical swatches.

Properties

Optional.

Default

center,center

Example

align=left,top

Swatches.buttonsnapmode

[Swatches.|<containerId>_swatches.]buttonsnapmode=snapin|snapout|overlay

<i>snapin</i>	Causes the buttons to align next to the swatches.
<i>snapout</i>	Causes the buttons to align next to the component border.
<i>overlay</i>	Causes the buttons to render on top of the swatches.

Properties

Optional.

Default

snapout

Example

buttonsnapmode=overlay

Swatches.direction

[Swatches.|<containerId>_swatches.]direction=auto|left|right

auto left right	Specifies the way swatches fill in the view. left sets left-to-right fill order; right reverses the order so that the view is filled in from right-to-left, and top-to-bottom. When auto is set, the component applies right mode when locale is set to ja; otherwise, left is used.
-----------------	---

Properties

Optional.

Default

auto

Example

direction=right

Swatches.enabledragging

[Swatches.|<containerId>_swatches.]enabledragging=0|1[,overdragvalue]

0 1	Enables or disables the ability for a user to scroll the swatches with a mouse or by using touch gestures
overdragvalue	Functions within the 0–1 range. It is a % value for movement in the wrong direction of the actual speed. If it is set to 1, it moves with the mouse. If it is set to 0, it does not let you move in the wrong direction at all.

Properties

Optional.

Default

1,0.5

Example

enabledragging=0

Swatches.fmt

[Swatches.|<containerId>_swatches.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha

jpg jpeg png png-alpha gif gif-alpha	Specifies the image format that the component uses for loading images from Image Server. If the specified format ends with -alpha, the component renders images as transparent content. For all other image formats the component treats images as opaque. Note that the component has a white background by default. Therefore, to make the background transparent set the background-color CSS property to transparent.
--------------------------------------	---

Properties

Optional.

Default


jpeg

Example

fmt-png-alpha

Swatches.iscommand

[Swatches.|<containerId>_swatches.]iscommand=*isCommand*

<i>isCommand</i>	<p>The Image Serving command string that is applied to all swatches. If it is specified in the URL, be sure that you HTTP-encode all occurrences of & and = as %26 and %3D, respectively.</p> <p> Note: Image sizing manipulation commands are not supported.</p>
------------------	---

Properties

Optional.

Default

None.

Example

When specified in the viewer URL.

iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000

When specified in the configuration data.

iscommand=op_sharpen=1&op_colorize=0xff0000

Swatches.maxloadradius

[Swatches.|<containerId>_swatches.]maxloadradius=-1|0|preloadnbr

-1 0 preloadnbr	<p>Specifies the component preload behavior. When set to -1 all swatches are loaded simultaneously when the component is initialized or the asset is changed.</p> <p>When set to 0 only visible swatches are loaded.</p> <p>preloadnbr defines how many invisible rows/columns around the visible area are preloaded.</p>
-----------------	---

Properties

Optional.

Default

1

Example

maxloadradius=-1

Swatches.pagemode

[Swatches.|<containerId>_swatches.]pagemode=0|1

0 1	<p>When toggled, the scroll buttons automatically cause the swatches to jump a full page length.</p> <p>Extra whitespace is shown on the last page if the swatches do not fit. Also, the last page has the same number of cells as any previous page.</p> <p>The scrollstep is ignored and mouse scrolling settles only on full pages.</p>
-----	--

Properties

Optional.

Default

0

Example

pagemode=1

Swatches.partialswatches

[Swatches. | <containerId>_swatches.]partialswatches=0 | 1

0 1	Specifies whether the component allows scrolling to stop when any of the swatches are partially visible (scrolling is not aligned). The recommended value is <code>false</code> or 0.
-------	---

Properties

Optional.

Default

0

Example

partialswatches=1

Swatches.scrollstep

[Swatches. | <containerId>_swatches.]scrollstep=*hStep*,*vStep*

<i>hStep</i>	Horizontal step.
<i>vStep</i>	Vertical step.

Specifies the number of swatches to scroll for each click or tap of the corresponding scroll button.

Properties

Optional.

Default

3, 3

Example

scrollstep=1,1

Swatches.textpos

[Swatches. | <containerId>_swatches.]textpos=bottom|top|left|right|none|tooltip

[Swatches. | <containerId>_colorSwatches.]textpos=bottom|top|left|right|none|tooltip

bottom top left right none tooltip	Specifies where the label is drawn relative to the swatch thumbnail image. That is, the label is centered at the specified location relative to the swatch thumbnail. When <code>tooltip</code> is specified, no label is drawn.
------------------------------------	--

Properties

Optional.

Default

none

Example

```
textpos=tooltip
```

VideoPlayer.autoplay

```
[VideoPlayer.|<containerId>_videoPlayer.]autoplay=0/1
```

<i>0/1</i>	Indicates whether the viewer starts playing the video on load. Some systems, like certain mobile devices, do not support AutoPlay.
------------	--

Properties

Optional.

Default

0

Example

```
autoplay=1
```

VideoPlayer.iconeffect

```
[VideoPlayer.|<containerId>_videoPlayer.]iconeffect=0/1[,count][,fade][,autoHide]
```

<i>0/1</i>	Enables the IconEffect to display on top of the video when the video is paused. On some devices native controls are used. In such case, the <code>iconeffect</code> modifier is ignored.
<i>count</i>	Specifies the maximum number of times the IconEffect appears and reappears. A value of -1 indicates that the icon reappears indefinitely.
<i>fade</i>	Specifies the duration of show or hide animation, in seconds.
<i>autoHide</i>	Sets number of seconds that the IconEffect stays visible before it auto-hides. That is, the time after fade-in animation is completed and before fade-out animation starts. A setting of 0 disables auto-hide behavior.

Properties

Optional.

Default

1,-1,0.3,0

Example

iconeffect=0

VideoPlayer.initialbitrate

[VideoPlayer.<containerId>_videoPlayer.]initialbitrate=value

value	<p>Sets the video bitrate—in kbits per seconds or kbps—that is used for initial playback of video on desktops.</p> <p>If this bitrate value does not exist in the Adaptive Video Set, then the video player starts the video that has the next lowest bitrate.</p> <p>If set to 0 the video player starts from the lowest possible bitrate. Applicable only for systems that do not have native support for HTML5 HLS video (which are Firefox, Chrome and Internet Explorer 11 browsers on Windows 10), and for Flash-enabled desktop systems, and when playback mode is set to <code>auto</code>.</p>
-------	---

Properties

Optional.

Default

1400

Example

initialbitrate=600

VideoPlayer.loop

[VideoPlayer.<containerId>_videoPlayer.]loop=0|1

0 1	Indicates whether the media plays again after playback is completed.
-----	--

Properties

Optional.

Default

0

Example

loop=1

VideoPlayer.playback

Configuration attribute for Mixed Media Video Viewer.

```
[VideoPlayer.<containerId>_videoPlayer.]playback=auto|progressive|flash
```

auto progressive flash	<p>Sets the type of playback used by the viewer. When <code>auto</code> is set, on most desktop browsers and all iOS devices, the viewer uses HTML5 streaming video in HLS format. It falls back to progressive HTML5 playback on certain systems like older Internet Explorer and Android.</p> <p>If <code>progressive</code> is specified, the viewer relies only on HTML5 playback as natively supported by browsers and plays video progressively on all systems.</p> <p>If <code>flash</code> is specified, the viewer uses Flash video playback if the Flash Player is installed; otherwise, HTML5 playback is used.</p> <p>For more information on the playback selection in auto and progressive modes please consult the Viewer SDK User Guide.</p>
------------------------	--

Properties

Optional.

Default

auto

Example

```
playback=progressive
```

VideoPlayer.posterimage

```
[VideoPlayer.<containerId>_videoPlayer.]posterimage=none|[image_id][?isCommands]
```

none [image_id][?isCommands]	<p>The image to display on the first frame before the video starts playing, resolved against <code>serverurl</code>. If specified in the URL, HTTP-encode the following:</p> <ul style="list-style-type: none"> • <code>?</code> as <code>%3F</code> • <code>&</code> as <code>%26</code> • <code>=</code> as <code>%3D</code> <p>If the <code>image_id</code> value is omitted, the component attempts to use the default poster image for that asset instead.</p> <p>When the video is specified as a path, the default poster images catalog id is derived from the video path as the <code>catalog_id/image_id</code> pair where <code>catalog_id</code> corresponds to the first token in the path and <code>image_id</code> is the name of the video with the extension removed. If the image with that ID does not exist, the poster image is not shown.</p>
------------------------------	--

	To prevent the display of the default poster image, specify <code>none</code> as the poster image value. If only the <i>isCommands</i> are specified the commands are applied to the default poster image before the image is displayed.
--	--

Properties

Optional.

Default

None.

Example

```
posterimage=none
```

VideoPlayer.progressivebitrate

```
[VideoPlayer.<containerId>_videoPlayer.]progressivebitrate=value
```

<i>value</i>	Specifies (in kbits per seconds or kbps) the desired video bit rate to play from an Adaptive Video Set in case the current system does not support adaptive video playback. The component picks up the video stream with the closest possible (but not exceeding) bitrate to the specified value. If all video streams in the Adaptive Video Set have higher quality than the specified value, the logic chooses the bitrate with the lowest quality.
--------------	--

Properties

Optional.

Default

700

Example

```
progressivebitrate=600
```

VideoPlayer.singleclick

```
[VideoPlayer.<containerId>_videoPlayer.]singleclick= none/playPause
```

<i>none/playPause</i>	Configures the mapping of single-click/tap to toggle play/pause. Setting to <code>none</code> disables single-click/tap to play/pause. If set to <code>playPause</code> , clicking the video toggles between playing and pausing the video. On some devices, you can use native controls. In such case, <code>singleclick</code> behavior is disabled.
-----------------------	--

Properties

Optional.

Default

playPause

Example

singleclick=none

VideoPlayer.smoothing

[VideoPlayer.<containerId>_videoPlayer.]smoothing=0|1

0 1	Specifies whether the video is smoothed (interpolated) when it is scaled. For smoothing to work, make sure the runtime is in the default high-quality mode. The default value is 0 (no smoothing). Set this property to 1 to take advantage of mipmapping image optimization. For Flash playback only.
-----	--

Properties

Optional.

Default

0

Example

smoothing=1

VideoPlayer.ssl

Configuration attribute for Mixed Media Video Viewer.



Note: This configuration attribute only applies to AEM 6.2 with installation of [Feature Pack NPR-13480](#) and to AEM 6.1 with installation of [Feature Pack NPR-15011](#).

[VideoPlayer.<containerId>_videoPlayer.]ssl=auto|on

auto on	<p>Controls whether the video is delivered over a secure SSL connection (HTTPS) or an insecure connection(HTTP).</p> <p>When set to <code>auto</code> the video delivery protocol is inherited from the protocol of the embedding web page. If the web page is loaded over HTTPS, the video is also delivered over HTTPS, and vice versa. If the web page is on HTTP, the video is delivered over HTTP.</p> <p>When set to <code>on</code>, video delivery always occurs over a secure connection without regard to the web page protocol.</p> <p>Affects published video delivery only and is ignored for video preview in Author mode.</p>
---------	--

Properties

Optional.

Default

auto

Example

```
ssl=on
```

See also [Secure Video Delivery](#).

VideoPlayer.waiticon

Configuration attribute for Mixed Media Video Viewer.

```
[VideoPlayer.|<containerId>_videoPlayer.]waiticon=0|1
```

0 1	Enables or disables buffering animation (wait icon) display.
-----	--

Properties

Optional.

Default

1

Example

```
waiticon=0
```

VideoScrubber.showtime

```
[VideoScrubber.|<containerId>_videoScrubber.]showtime=0|1
```

0 1	Enables or disables the time played bubble when adjusting knob position.
-----	--

Properties

Optional.

Default

1

Example

```
showtime=0
```

VideoScrubber.timepattern

[VideoScrubber.|<containerId>_videoScrubber.]timepattern=[h:]m|mm:s|ss

[h:]m mm:s ss	<p>Sets the pattern for the time that is displayed in the time bubble, where h is hours, m is minutes, and s is seconds.</p> <p>The number of letters used for each time unit determines the number of digits to display for the unit. If the number cannot fit into the given digits, the equivalent value is displayed in the subsequent unit.</p> <p>For example, if the current movie time is 67 minutes and 5 seconds, the time pattern m:ss shows as 67:05. The same time is displayed as 1:07:5 if the given time pattern is h:mm:s.</p>
---------------	---

Properties

Optional.

Default

m:ss

Example

timepattern=h:mm:ss

VideoTime.timepattern

[VideoTime.|<containerId>_videoTime.]timepattern=[h:]m|mm:s|ss

[h:]m mm:s ss	<p>Sets the pattern for the time that is displayed in the control bar, where h is hours, m is minutes, and s is seconds.</p> <p>The number of letters used for each time unit determines the number of digits to display for the unit. If the number cannot fit into the given digits, the equivalent value is displayed in the subsequent unit.</p> <p>For example, if the current movie time is 67 minutes and 5 seconds, the time pattern m:ss shows as 67:05. The same time is displayed as 1:07:5 if the given time pattern is h:mm:s.</p>
---------------	---

Properties

Optional.

Default

m:ss

Example

timepattern=h:mm:ss

zoomMode

Sets the type of zoom interaction.

zoomMode=continuous | inline | auto

continuous inline auto	<p>continuous enables classic zoom where the image gradually zooms in as you click, double-tap, or pinch out in the main view. You need to explicitly zoom out or reset the zoom state to get back to the initial view.</p> <p>inline enables instant zoom, where the zoomed image instantly appears as you hover the main view on desktop or to touch and hold on a touch device; image automatically reverts to initial state once you move your mouse from the view or release your finger. Note that in inline mode nested image sets are flattened and displayed as individual thumbnails. auto activates inline mode on desktop and continuous mode on touch devices.</p>
----------------------------	---

Properties

Optional.

Default

reset on desktop computers; zoomReset on touch devices.

Example

doubleclick=zoom

ZoomView.doubleclick

[ZoomView.<containerId>_zoomView.]doubleclick=none | zoom | reset | zoomReset

none zoom reset zoomReset	<p>Configures the mapping of double-click/tap to zoom actions. Setting to none disables double-click/tap zoom. If set to zoom clicking the image zooms in one zoom step; CTRL+Click zooms out one zoom step. Setting to reset causes a single click on the image to reset the zoom to the initial zoom level. For zoomReset, reset is applied if the current zoom factor is at or beyond the specified limit, otherwise zoom is applied.</p>
---------------------------------	--

Properties

Optional.

Default

reset on desktop computers; zoomReset on touch devices.

Example

```
doubleclick=zoom
```

ZoomView.enableHD

```
[ZoomView.|<containerId>_zoomView.|enableHD=always|never|limit[,number]
```

<code>always never limit</code>	<p>Enable, limit or disable optimization for devices where <code>devicePixelRatio</code> is greater than 1, that is devices with high-density display like iPhone4 and similar devices. If active then the component limits the size of the IS image request as if the device only had a pixel ratio of 1 and that way reducing the bandwidth.</p> <p>See Example 2 below.</p>
<code>number</code>	<p>If using the limit setting, the component enables high pixel density only up to the specified limit.</p> <p>See Example 2 below.</p>

Properties

Optional.

Default

```
limit,1500
```

Example

The following are the expected results when you use this configuration attribute with the viewer, and the viewer size is 1000 x 1000:

If the set variable equals	Result
<code>always</code>	<p>The pixel density of the screen/device is always taken into account.</p> <ul style="list-style-type: none"> • If the screen pixel density = 1, then the requested image is 1000 x 1000. • If the screen pixel density = 1.5, then the requested image is 1500 x 1500. • If the screen pixel density = 2, then the requested image is 2000 x 2000.
<code>never</code>	<p>It always uses a pixel density of 1 and ignores the device's HD capability. Therefore, the requested image requested is always 1000 x 1000.</p>
<code>limit<number></code>	<p>A device pixel density is requested and served only if the resulting image is below the specified limit.</p> <p>The limit number applies to either the width or the height dimension.</p>

If the set variable equals	Result
	<ul style="list-style-type: none"> • If the limit number is 1600, and the pixel density is 1.5, then the 1500 x 1500 image is served. • If the limit number is 1600, and the pixel density is 2, then the 1000 x 1000 image is served because the 2000 x 2000 image exceeds the limit. <p>Best practice: The limit number needs to work in conjunction with the company setting for maximum size image. Therefore, set the limit number to equal the company maximum image size setting.</p>

ZoomView.fmt

[ZoomView.|<containerId>_zoomView.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha

jpg jpeg png png-alpha gif gif-alpha	Specifies the image format that the component uses for loading images from Image Server. If the specified format ends with <code>-alpha</code> , the component renders images as transparent content. For all other image formats the component treats images as opaque. The component has a white background by default. Therefore, to make it transparent, set the <code>background-color</code> CSS property to transparent.
--------------------------------------	---

Properties

Optional.

Default

jpeg

Example

fmt=png-alpha

ZoomView.iconeffect

[ZoomView.|<containerId>_zoomView.]iconeffect=0|1[,count][,fade][,autoHide]

0 1	Enables the <code>iconeffect</code> to display on the top of the image when the image is in a reset state and it is suggestive of an available action to interact with the image.
<i>count</i>	Specifies the maximum number of times the <code>iconeffect</code> appears and reappears. A value of <code>-1</code> indicates that the icon always reappears indefinitely.
<i>fade</i>	Specifies the duration of the show or hide animation, in seconds.

<i>autoHide</i>	Sets the number of seconds that the <code>iconeffect</code> stays fully visible before it auto hides. That is, the time after the fade-in animation is completed but before the fade out animation starts. A setting of 0 disables the auto-hide behavior.
-----------------	--

Properties

Optional.

Default


1,1,0.3,3

Example

`iconeffect=0`

ZoomView.iscommand

`[ZoomView.<containerId>_zoomView.iscommand=isCommand`

<i>iscommand</i>	<p>The Image Serving command string that is applied to zoom image. If specified in the URL all occurrences of & and = must be HTTP-encoded as %26 and %3D, respectively.</p> <p> Note: Image sizing manipulation commands are not supported.</p>
------------------	--

Properties

Optional.

Default

None.

Example

When specified in the viewer URL:

`iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000`

When specified in the config data:

`iscommand=op_sharpen=1&op_colorize=0xff0000`

ZoomView.reset

`[ZoomView.<containerId>_zoomView.reset=0|1`

0 1	Resets the view port when frame (image) changes. If set to 0 it preserves the current view port with the best possible fit while preserving the aspect ratio of the newly set image.
-----	--

Properties

Optional.

Default

1

Example

```
reset=0
```

ZoomView.rgn

```
[ZoomView. |<containerId>_zoomView. ]rgn=x,y,w,h
```

x,y,w,h	Initial region of interest in pixel coordinates. If this is not specified, the entire image is fit within the initial viewport.
---------	---

Properties

Optional.

Default

None.

Example

```
rgn=0,0,500,500
```

ZoomView.rgnN

```
[ZoomView. |<containerId>_zoomView. ]rgnN=x,y,w,h
```

x,y,w,h	Initial region of interest in normalized coordinates. If this is not specified, the entire image is fit within the initial viewport.
---------	--

Properties

Optional.

Default

None.

Example

```
rgnN=0,0,0.5,0.5
```

ZoomView.singleclick

[ZoomView. | <containerId>_zoomView.]singleclick=*none* | *zoom* | *reset* | *zoomReset*

<i>none</i> <i>zoom</i> <i>reset</i> <i>zoomReset</i>	Configures the mapping of single-click/tap to zoom actions. Setting to <i>none</i> disables single-click/tap zoom. If set to <i>zoom</i> clicking the image zooms in one zoom step; CTRL+Click zooms out one zoom step. Setting to <i>reset</i> causes a single click on the image to reset the zoom to the initial zoom level. For <i>zoomReset</i> , reset is applied if the current zoom factor is at or beyond the specified limit, otherwise zoom is applied.
---	--

Properties

Optional.

Default

zoomReset on desktop computers; *none* on touch devices.

Example

singleclick=zoom

ZoomView.transition

[ZoomView. | <containerId>_zoomView.]transition=*time*[,*easing*]

<i>time</i>	Specifies the time in seconds that the animation for a single zoom step action takes.
<i>easing</i>	<p>Creates an illusion of acceleration or deceleration which makes the transition appear more natural. You can set easing to one of the following:</p> <ul style="list-style-type: none">• 0 (auto)• 1 (linear)• 2 (quadratic)• 3 (cubic)• 4 (quartic)• 5 (quintic) <p>Auto mode always uses linear transition when elastic zoom is disabled (default). Otherwise, it fits one of the other easing functions based on the transition time. That is, the shorter the transition time the higher the easing function is used to accelerate the acceleration or deceleration effect.</p>

Properties

Optional.

Default

0.5,0

Example

transition=2,2

ZoomView.zoomstep

[ZoomView.|<containerId>_zoomView.]zoomstep=step[,limit]

<i>step</i>	Configures the number zoom in and zoom out actions that are required to increase or decrease the resolution by a factor of two. The resolution change for each zoom action is 2 ¹ per step. Set to 0 to zoom to full resolution with a single zoom action.
<i>limit</i>	Specifies the maximum zoom resolution, relative to the full resolution image. The default is 1.0, which does not allow zooming beyond full resolution.

Properties

Optional.

Default

1,1

Example

zoomstep=2,3

Javascript API reference for Mixed Media Viewer

The main class of the Mixed Media Viewer is `MixedMediaViewer`. It is declared in the `s7viewers` namespace. This JavaScript API covers constructor, methods, and call backs of this particular class.

In all the following examples, <instance> stands for the actual name of the JavaScript viewer object that is instantiated from the `s7viewers.MixedMediaViewer` class.

dispose

JavaScript API reference for Mixed Media Viewer.

`dispose()`

Disposes this viewer instance by releasing all resources used by the viewer logic and deleting all inner objects and components created by the viewer in runtime.

The web page code should also delete the viewer instance variable as well to completely remove the viewer from the web browser memory.

If the web page code has registered event listeners directly on Viewer SDK components used by the viewer—or stored external references to such components—such listeners must be explicitly unregistered by the web page code, and such external component references must be deleted prior to calling `dispose()`.

Do not access the Viewer API any more after `dispose()` is called.

Parameters

None.

Returns

None.

Example

```
<instance>.dispose()
```

getComponent

JavaScript API reference for Mixed Media Viewer

```
getComponent(componentId)
```

Returns a reference to the Viewer SDK component that is used by the viewer. The web page can use this method to extend or customize the behavior of the out-of-box viewer. Call this method only after the `initComplete` viewer callback has run, otherwise the component may not be created yet by the viewer logic.

Parameters

componentID – {String} an ID of the Viewer SDK component used by the viewer. This viewer supports the following component IDs:

Component ID	Viewer SDK component class name
parameterManager	s7sdk.ParameterManager
container	s7sdk.common.Container
mediaSet	s7sdk.set.MediaSet
zoomView	s7sdk.image.ZoomView
flyoutZoomView	s7sdk.image.FlyoutZoomView
spinView	s7sdk.set.SpinView
videoPlayer	s7sdk.video.VideoPlayer
controls	s7sdk.common.ControlBar
videoScrubber	s7sdk.video.VideoScrubber
videoTime	s7sdk.video.VideoTime

Component ID	Viewer SDK component class name
closedCaptionButton	s7sdk.common.ClosedCaptionButton
swatches	s7sdk.set.Swatches
colorSwatches	s7sdk.set.Swatches
zoomInButton	s7sdk.common.ZoomInButton
zoomOutButton	s7sdk.common.ZoomOutButton
zoomResetButton	s7sdk.common.ZoomResetButton
spinLeftButton	s7sdk.common.PanLeftButton
spinRightButton	s7sdk.common.PanRightButton
mutableVolume	s7sdk.video.MutableVolume
playPauseButton	s7sdk.common.PlayPauseButton
fullScreenButton	s7sdk.common.FullScreenButton
closeButton	s7sdk.common.CloseButton

When working with SDK APIs it is important to use correct, fully qualified SDK namespace as described in [Localization of user interface elements](#).

See the Viewer SDK API documentation for more information about a particular component.

Returns

{Object} a reference to Viewer SDK component. The method returns `null` if the `componentId` is not a supported viewer component or if the component was not yet created by the viewer logic.

Example

init

JavaScript API reference for Mixed Media Viewer.

```
init()
```

Starts the initialization of the Mixed Media Viewer. By this time the container DOM element must be created so that the viewer code can find it by its ID.

If the container element is not a part of the web page layout just yet (for example, it may be hidden using `display:none` style assigned to it), the viewer suspends its initialization process until the moment when the web page brings the container element back to the layout. When this occurs, the viewer load automatically resumes.

Only call this method once during the viewer life cycle; subsequent calls are ignored.

Parameters

None.

Returns

{Object} A reference to the viewer instance.

Example

```
<instance>.init()
```

MixedMediaViewer

JavaScript API reference for Mixed Media Viewer.

```
MixedMediaViewer([config])
```

Constructor, creates a new Mixed Media Viewer instance.

Parameters

<i>config</i>	<p>{Object} optional JSON configuration object, allows all the viewer settings to pass to the constructor and avoid calling individual setter methods. Contains the following properties:</p> <ul style="list-style-type: none">• containerId – {String} ID of the DOM container (normally a <code>DIV</code>) that the viewer is inserted into. It is not necessary to have the container element created by the time this method is called. However, the container must exist when <code>init()</code> is run. Required.• params – {Object} JSON object with viewer configuration parameters where the property name is either a viewer-specific configuration option or an SDK modifier, and the value of that property is a corresponding settings value. Required.• handlers – {Object} JSON object with viewer event callbacks, where the property name is the name of supported viewer event, and the property value is a JavaScript function reference to an appropriate callback. Optional. See Event callbacks for more information about viewer events.• localizedTexts – {Object} JSON object with localization data. Optional. See Localization of user interface elements for more information. <p>See also the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
---------------	--

Returns

None.

Example

```
var mixedMediaViewer = new s7viewers.MixedMediaViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/Mixed_Media_Set_Sample",
    "serverurl": "http://s7dl.scene7.com/is/image/",
    "videoserverurl": "http://s7dl.scene7.com/is/content/"
  },
  "handlers": {
    "initComplete": function() {
      console.log("init complete");
    }
  },
  "localizedTexts": {
    "en": {
      "CloseButton.TOOLTIP": "Close"
    },
    "fr": {
      "CloseButton.TOOLTIP": "Fermer"
    }
  },
  defaultLocale: "en"
});
```

setAsset

JavaScript API reference for Mixed Media Viewer.

```
setAsset(asset[,data])
```

Sets the new asset and optional additional asset data. You can call this parameter at any time, either before or after `init()`. If it is called after `init()`, the viewer swaps the asset at runtime.

See also [init](#).

Parameters

asset – String} new asset ID or explicit mixed media set, with optional Image Serving modifiers appended after `?`.

Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.

data – {JSON} location of the new caption file.

If not specified, the caption button is not visible in the user interface. Captions specified with this parameter apply to the video that comes first in the mixed media set; subsequent videos play without captions. This viewer supports the following component IDs:

Component ID	Viewer SDK component class name
posterimage	Image to display on the first frame before the video starts playing. See VideoPlayer.posterimage .
caption	Location of the new caption file. If not specified, the caption button is not visible in the user interface. Captions specified with this parameter apply to the video that comes first in the media set. Subsequent videos play without captions.

Returns

None.

Example

Single media set reference:

```
<instance>.setAsset("Scene7SharedAssets/Mixed_Media_Set_Sample")
```

Explicit media set:

```
<instance>.setAsset("Scene7SharedAssets/Mixed_Media_Set_Sample", "Scene7SharedAssets/Mixed_Media_Set_Sample")
```

Sharpening modifier added to all images in the set:

```
<instance>.setAsset("Scene7SharedAssets/Mixed_Media_Set_Sample?op_sharpen=1")
```

setContainerId

JavaScript API reference for Mixed Media Viewer.

```
setContainerId(containerId)
```

Sets the ID of the DOM container (normally a DIV) into which the viewer is inserted. It is not necessary to have the container element created by the time this method is called. However, the container must exist when `init()` is run. It must be called before `init()`. This method is optional if viewer configuration information is passed with `config` JSON object to the constructor.

<i>containerId</i>	{string} ID of container.
--------------------	---------------------------

Returns

None.

Example

```
<instance>.setContainerId("s7viewer");
```

setHandlers

JavaScript API reference for Mixed Media Viewer.

```
setHandlers(handlers)
```

Specifies zero or more callback handlers. A call to this method fully overwrites event handlers that were previously assigned for that viewer instance. Must be called before `init()`.

Parameter

<i>handlers</i>	<p>{Object} JSON object with viewer event callbacks, where the property name is the name of the supported viewer event and the property value is a JavaScript function reference to an appropriate callback.</p> <p>See Event callbacks for more information about viewer events.</p>
-----------------	---

Returns

None.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    console.log("init complete");
  }
})
```

setLocalizedTexts

JavaScript API reference for Mixed Media Viewer.

setLocalizedTexts(*localizationInfo*)

<i>localizationInfo</i>	<p>{Object} JSON object with localization data.</p> <p>See Localization of user interface elements for more information.</p> <p>See also the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
-------------------------	--

Sets localization SYMBOL values for one or more locales. This parameter must be called before `init()`.

See also [init](#).

Returns

None.

Example

```
<instance>.setLocalizedTexts({"en":{"CloseButton.TOOLTIP":"Close"},"fr":{"CloseButton.TOOLTIP":"Fermer"},defaultLocale:"en"})
```

setParam

JavaScript API reference for Mixed Media Viewer.

setParam(*name*, *value*)

Sets the viewer parameter to a specified value. The parameter is either a viewer-specific configuration option or a software development kit modifier. This parameter is called before `init()`. This method is optional if viewer configuration information is passed with `config` JSON object to the constructor.

See also [init](#).

<i>name</i>	{string} name of parameter.
<i>value</i>	{string} value of parameter. The value cannot be percent-encoded.

Returns

None.

Example

```
<instance>.setParam("style", "customStyle.css")
```

setParams

JavaScript API reference for Mixed Media Viewer.

```
setParams(params)
```

Sets one or more parameters to a given value. The method argument syntax is identical to a URL query string. That is, it represents name=value pairs separated with &. Just as in a query string, the names and values are percent-encoded using UTF8. Before you call `init()`, this parameter must be called. This method is optional if viewer configuration information is passed with `config` JSON object to the constructor.

See also [init](#).

<i>params</i>	{string} name=value parameter pairs separated with &.
---------------	---

Returns

None.

Example

```
<instance>.setParams("ZoomView.zoomstep=2,3&ZoomView.iscommand=op_sharpen%3d1")
```

Event callbacks

The viewer supports JavaScript event callbacks that the web page uses to track the viewer initialization process or runtime behavior.

Callback handlers are assigned by passing event names and corresponding handler functions with the `handlers` property to `config` JSON object in the viewer's constructor. Alternatively, it is possible to use `setHandlers()` API method.

Supported viewer events include the following:

- `initComplete` – triggers when viewer initialization is complete and all internal components are created, so that it is possible to use `getComponent()` API. The callback handler does not take any arguments.
- `trackEvent` – triggers each time an event occurs inside the viewer which may be handled by an event tracking system, such as Adobe Analytics. The callback handler takes the following arguments:
 - `objID` {String} not currently used.
 - `compClass` {String} not currently used.
 - `instName` {String} an instance name of the Viewer SDK component that triggered the event.
 - `timeStamp` {Number} event time stamp.
 - `eventInfo` {String} event payload.

See also [MixedMediaViewer](#) and [setHandlers](#).

Customizing Mixed Media Viewer

All visual customization and most behavior customization for the Mixed Media Viewer is done by creating a custom CSS.

The suggested workflow is to take the default CSS file for the appropriate viewer, copy it to a different location, customize it, and specify the location of the customized file in the `style=` command.

Default CSS files can be found at the following location:

```
<s7_viewers_root>/html5/MixedMediaViewer_light.css
```

The viewer is supplied with two out-of-the-box CSS files, for "light" and "dark" color schemes. The "light" version is used by default, but you can switch to the "dark" version by using the following standard CSS:

```
<s7_viewers_root>/html5/MixedMediaViewer_dark.css
```

Custom CSS file must contain the same class declarations as the default one. If a class declaration is omitted, the viewer does not function properly because it does not provide built-in default styles for user interface elements.

Alternative way to provide custom CSS rules is to use embedded styles directly on the web page or in one of linked external CSS rules.

When creating custom CSS keep in mind that the viewer assigns `.s7mixedmediaviewer` class to its container DOM element. If you are using external CSS file passed with `style=` command, use `.s7mixedmediaviewer` class as parent class in descendant selector for your CSS rules. If you are doing embedded styles on the web page, additionally qualify this selector with an ID of the container DOM element as follows:

```
#<containerId>.s7mixedmediaviewer
```

Building responsive designed CSS

It is possible to target different devices and embedding sizes in CSS to make your content display differently, depending on a user's device or a particular web page layout. This includes, but is not limited to, different web page layouts, user interface element sizes, and artwork resolution.

The viewer supports two methods for creating responsive designed CSS: CSS markers and standard CSS media queries. You can use these methods independently or together.

CSS markers

To assist in creating responsive designed CSS, the viewer supports CSS markers which special CSS classes dynamically assigned to the top-level viewer container element based on the run-time viewer size and the input type used on the current device.

The first group of CSS markers includes `.s7size_large`, `.s7size_medium`, and `.s7size_small` classes. They are applied based on the runtime area of the viewer container. That is, if the viewer area is equal to or bigger than the size of a common desktop monitor `.s7size_large` is used; if the area is close in size to a common tablet device `.s7size_medium` is assigned. For areas similar to mobile phone screens `.s7size_small` is set. The primary purpose of these CSS markers is to create different user interface layouts for different screens and viewer sizes.

The second group of CSS Markers includes `.s7mouseinput` and `.s7touchinput`. `.s7touchinput` is set if the current device has touch input capabilities; otherwise, `.s7mouseinput` is used. These markers are intended to create user interface input elements with different screen sizes for different input types, because normally touch input requires larger elements. In case the device has both mouse input and touch capabilities, `.s7touchinput` is set and the viewer renders a touch-friendly user interface.

The following sample CSS sets the zoom in button size to 28 x 28 pixels on systems with mouse input, and 56 x 56 pixels on touch devices. In addition, it hides the button completely if the viewer size becomes really small:

```
.s7mixedmediaviewer.s7mouseinput .s7zoominbutton {
  width:28px;
  height:28px;
}
.s7mixedmediaviewer.s7touchinput .s7zoominbutton {
```

```

    width:56px;
    height:56px;
}
.s7mixedmediaviewer.s7size_small .s7zoominbutton {
    visibility:hidden;
}

```

To target devices with a different pixel density, use CSS media queries. The following media query block would contain CSS that is specific to high density screens:

```

@media screen and (-webkit-min-device-pixel-ratio: 1.5)
{
}

```

Using CSS markers is the most flexible way of building responsive designed CSS as it allows you to target not only device screen size but actual viewer size, which may be useful for responsive design page layouts.

Use the default viewer CSS file as an example of a CSS markers approach.

CSS media queries

Device sensing can also be done using pure CSS media queries. Everything enclosed within a given media query block is applied only when it is run on a corresponding device.

When applied to Mobile Viewers, use four CSS media queries, defined in your CSS in the following order:

1. Contains only rules specific for all touch devices.

```

@media only screen and (max-device-width:13.5in) and (max-device-
height:13.5in) and (max-device-width:799px),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in)
and (max-device-height:799px)
{
}

```

2. Contains only rules specific for tablets with high resolution screens.

```

@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px) and (-webkit-min-device-pixel-ratio:1.5),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px) and (-webkit-min-device-pixel-ratio:1.5)
{
}

```

3. Contains only rules specific for all mobile phones.

```

@media only screen and (max-device-width:9in) and (max-device-height:9in)
{
}

```

4. Contains only rules specific for mobile phones with high resolution screens.

```

@media only screen and (max-device-width:9in) and (max-device-height:9in) and
(-webkit-min-device-pixel-ratio: 1.5),
only screen and (device-width:720px) and (device-height:1280px) and
(-webkit-device-pixel-ratio: 2),
only screen and (device-width:1280px) and (device-height:720px) and
(-webkit-device-pixel-ratio: 2)
{
}

```

Using a media queries approach, you should organize CSS with device sensing as follows:

- First, the desktop-specific section defines all properties that are either desktop-specific or common to all screens.

- And second, the four media queries go in the order defined above and provide CSS rules that are specific for the corresponding device type.

There is no need to duplicate the entire viewer CSS in each media query. Only properties that are specific to given devices are redefined inside a media query.

CSS Sprites

Many viewer user interface elements are styled using bitmap artwork and have more than one distinct visual state. A good example is a button that normally has at least three different states: "up", "over", and "down". Each state requires its own bitmap artwork assigned.

With a classic approach to styling, the CSS would have a separate reference to individual image file on the server for each state of the user interface element. The following is a sample CSS for styling a zoom-in button:

```
.s7mixedmediaviewer.s7mouseinput .s7zoominbutton[state='up'] {
background-image:url(images/v2/ZoomInButton_dark_up.png);
}
.s7mixedmediaviewer.s7mouseinput .s7zoominbutton[state='over'] {
background-image:url(images/v2/ZoomInButton_dark_over.png);
}
.s7mixedmediaviewer.s7mouseinput .s7zoominbutton[state='down'] {
background-image:url(images/v2/ZoomInButton_dark_down.png);
}
.s7mixedmediaviewer.s7mouseinput .s7zoominbutton[state='disabled'] {
background-image:url(images/v2/ZoomInButton_dark_disabled.png);
}
```

The drawback to this approach is that the end user experiences flickering or delayed user interface response when the element is interacted with for the first time. This action occurs because the image artwork for the new element state is not yet downloaded. Also, this approach may have a slight negative impact on performance because of an increase in the number of HTTP calls to the server.

CSS sprites is a different approach where image artwork for all element states is combined into a single PNG file called a "sprite". Such "sprite" has all visual states for the given element positioned one after another. When styling a user interface element with sprites the same sprite image is referenced for all different states in the CSS. Also, the `background-position` property is used for each state to specify which part of the "sprite" image is used. You can structure a "sprite" image in any suitable way. Viewers normally have it vertically stacked. Below is a "sprite"-based example of styling the same zoom-in button from above:

```
.s7mixedmediaviewer .s7zoominbutton[state] {
background-image: url(images/v2/ZoomInButton_dark_sprite.png);
}
.s7mixedmediaviewer.s7mouseinput .s7zoominbutton[state='up'] {
background-position: -84px -560px;
}
.s7mixedmediaviewer.s7mouseinput .s7zoominbutton[state='over'] {
background-position: -56px -560px;
}
.s7mixedmediaviewer.s7mouseinput .s7zoominbutton[state='down'] {
background-position: -28px -560px;
}
.s7mixedmediaviewer.s7mouseinput .s7zoominbutton[state='disabled'] {
background-position: -0px -560px;
}
```

General styling notes and advice

- All paths to external assets within CSS are resolved against the CSS location, not the viewer HTML page location. Be aware of this rule when you copy the default CSS to a different location. Either copy the default assets as well or update paths within the custom CSS.

- The preferred format for bitmap artwork is PNG.
- Bitmap artwork is assigned to user interface elements using the `background-image` property.
- The `width` and `height` properties of a user interface element define its logical size. The size of the bitmap passed to `background-image` does not affect logical size.
- To use the high pixel density of high-resolution screens like Retina, specify bitmap artwork twice as large as the logical user interface element size. Then, apply the `-webkit-background-size:contain` property to scale the background down to the logical user interface element size.
- To remove a button from the user interface, add `display:none` to its CSS class.
- You can use various formats for color value that CSS supports. If you need transparency, use the format `rgba(R,G,B,A)`. Otherwise, you can use the format `#RRGGBB`.
- When customizing the viewer user interface with CSS the use of the `!IMPORTANT` rule is not supported to style viewer elements. In particular, `!IMPORTANT` rule should not be used to override any default or run-time styling provided by the viewer or Viewer SDK. The reason is that it may affect the behavior of proper components. Instead, you should use CSS selectors with the proper specificity to set CSS properties that are documented in this reference guide.

Common User Interface Elements

The following is user interface elements reference documentation that applies to Mixed Media Viewer:

Caption button

Toggles the closed caption display on and off. It is not visible if the caption parameter is not specified. You can use CSS to size, skin, and position this button relative to the control bar that contains it.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7closedcaptionbutton
```

CSS property	Description
<code>top</code>	Position from the top border, including padding.
<code>right</code>	Position from the right border, including padding.
<code>left</code>	Position from the left border, including padding.
<code>bottom</code>	Position from the bottom border, including padding.
<code>width</code>	Width of the button.
<code>height</code>	Height of the button.
<code>background-image</code>	The image that is displayed for a given button state.
<code>background-position</code>	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the *state* attribute selector and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the state when captions are visible and *selected='false'* is used when captions are hidden.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a closed caption button that is 28 x 28 pixels, positioned four pixels from the top and 68 pixels from the right edge of the control bar, and displays a different image for each of the four different button states when selected or not selected.

```
.s7mixedmediaviewer .s7closedcaptionbutton {
  position:absolute;
  top:4px;
  right:68px;
  width:28px;
  height:28px;
}
.s7mixedmediaviewer .s7closedcaptionbutton[selected='true'][state='up'] {
  background-image:url(images/v2/ClosedCaptionButton_up.png);
}
.s7mixedmediaviewer .s7closedcaptionbutton[selected='true'][state='over'] {
  background-image:url(images/v2/ClosedCaptionButton_over.png);
}
.s7mixedmediaviewer .s7closedcaptionbutton[selected='true'][state='down'] {
  background-image:url(images/v2/ClosedCaptionButton_down.png);
}
.s7mixedmediaviewer .s7closedcaptionbutton[selected='true'][state='disabled'] {
  background-image:url(images/v2/ClosedCaptionButton_disabled.png);
}
.s7mixedmediaviewer .s7closedcaptionbutton[selected='false'][state='up'] {
  background-image:url(images/v2/ClosedCaptionButton_disabled.png);
}
.s7mixedmediaviewer .s7closedcaptionbutton[selected='false'][state='over'] {
  background-image:url(images/v2/ClosedCaptionButton_over.png);
}
.s7mixedmediaviewer .s7closedcaptionbutton[selected='false'][state='down'] {
  background-image:url(images/v2/ClosedCaptionButton_down.png);
}
.s7mixedmediaviewer .s7closedcaptionbutton[selected='false'][state='disabled'] {
  background-image:url(images/v2/ClosedCaptionButton_disabled.png);
}
```

Close button

Clicking or tapping this button closes the containing web page. This button only appears if the `closebutton` parameter is set to 1. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7closebutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.

CSS property	Description
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a close button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7mixedmediaviewer .s7closebutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7mixedmediaviewer .s7closebutton [state='up'] {
background-image:url(images/v2/CloseButton_dark_up.png);
}
.s7mixedmediaviewer .s7closebutton [state='over'] {
background-image:url(images/v2/CloseButton_dark_over.png);
}
.s7mixedmediaviewer .s7closebutton [state='down'] {
background-image:url(images/v2/CloseButton_dark_down.png);
}
.s7mixedmediaviewer .s7closebutton [state='disabled'] {
background-image:url(images/v2/CloseButton_dark_disabled.png);
}
```

Color swatches

Color swatches consist of a row of thumbnail images with optional scroll buttons on the left and right hand side. Color swatches are only visible on the desktop if all thumbnails cannot fit into the width of the container. On mobile devices, or if thumbnails can fit into the container width, scroll buttons are not shown.

The appearance of the swatches container is controlled with the CSS class selector:

```
.s7mixedmediaviewer .s7colorswatches .s7swatches
```

CSS properties of the color swatches

width	The width of the swatches.
-------	----------------------------

height	The height of the swatches.
bottom	The vertical swatches offset relative to the viewer container.

Example – to set up swatches with a height of 100 pixels.

```
.s7mixedmediaviewer .s7colorswatches .s7swatches {
  height: 100px;
}
```

The spacing between the swatch thumbnails is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7thumbcell
```

CSS property	Description
margin	The size of horizontal and vertical margin around each thumbnail. Actual thumbnail spacing equals to the sum of left and right margin set for <code>.s7thumbcell</code> .

Example

To set spacing to ten pixels both vertically and horizontally.

```
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7thumbcell {
  margin: 5px;
}
```

The appearance of the individual thumbnail is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7thumb
```

CSS property	Description
width	Width of the thumbnail.
height	Height of the thumbnail.
border	Border of the thumbnail.



Note: Thumbnail supports the *state* attribute selector, which can be used to apply different skins to different thumbnail states. In particular, *state="selected"* corresponds to the thumbnail for the image that is currently displayed in the main view, *state="default"* corresponds to the rest of thumbnails, and *state="over"* is used on mouse hover.

Example – to set up thumbnails that are 56 x 56 pixels, have a light grey default border, and a dark grey selected border.

```
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7thumb {
  width: 56px;
  height: 56px;
}
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7thumb[state="default"] {
  border: 1px solid #dddddd;
}
```

```
.s7mixedviewer .s7colorswatches .s7swatches .s7thumb[state="selected"] {
  border: 1px solid #666666;
}
```

The appearance of the left and right scroll buttons are controlled with the following CSS class selectors:

```
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7scrollleftbutton
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7scrollrightbutton
```

It is not possible to position scroll buttons using CSS top, left, bottom, and right properties. Instead, the viewer logic positions them automatically.

CSS property	Description
width	Width of the scroll button.
height	Height of the scroll button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states: *up, down, over, and disabled*.

Example – to set up scroll buttons that are 56 x 56 pixels and have different artwork for each state.

```
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7scrollleftbutton {
  background-size: auto;
  width: 56px;
  height: 56px;
}
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7scrollleftbutton[state="up"]{
  background-image:url(images/v2/ScrollLeftButton_up.png);
}
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7scrollleftbutton[state="over"]{
  background-image:url(images/v2/ScrollLeftButton_over.png);
}
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7scrollleftbutton[state="down"]{
  background-image:url(images/v2/ScrollLeftButton_down.png);
}
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7scrollleftbutton[state="disabled"]{
  background-image:url(images/v2/ScrollLeftButton_disabled.png);
}
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7scrollrightbutton {
  background-size: auto;
  width: 56px;
  height: 56px;
}
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7scrollrightbutton[state="up"]{
  background-image:url(images/v2/ScrollRightButton_up.png);
}
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7scrollrightbutton[state="over"]{
  background-image:url(images/v2/ScrollRightButton_over.png);
}
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7scrollrightbutton[state="down"]{
  background-image:url(images/v2/ScrollRightButton_down.png);
}
```

```

}
.s7mixedmediaviewer .s7colorswatches .s7swatches .s7scrollrightbutton[state="disabled"]{
  background-image:url(images/v2/ScrollRightButton_disabled.png);
}

```

Control bar

The control bar is the rectangular area that contains and sits behind all the user interface controls available for the video viewer, such as the play/pause button, volume controls, and so on.

The control bar always takes the entire available viewer width. It is possible to change its color, height, and vertical position by CSS, relative to the video viewer container.

The following CSS class selector controls the appearance of the control bar:

```
.s7mixedmediaviewer .s7controlbar
```

CSS properties of the control bar

height	Height of the control bar.
background-color	Background color of the control bar.

Example

To set up a mixed media viewer with a gray control bar that is 30 pixels tall.

```

.s7mixedmediaviewer .s7controlbar {
  height: 30px;
  background-color: rgb(51, 51, 51);
}

```

Flyout Zoom View

In inline zoom mode main view consists of the static image, zoomed image shown in the flyout view over the static image and the tip message shown on top of static image.

CSS properties of the main viewer area

The appearance of the main view is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7flyoutzoomview
```

CSS property	Description
background-color	The background color of the main view.

Example – to make the main view transparent:

```

.s7mixedmediaviewer .s7flyoutzoomview {
  background-color: transparent;
}

```

The appearance of the tip message is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7flyoutzoomview .s7tip
```

It is possible to configure font style, size appearance, and vertical offset through CSS. However, horizontal alignment is managed by the viewer logic. Overriding it through CSS using `left` or `right` properties is not supported.

CSS properties of the tip message

CSS property	Description
<code>background-color</code>	Message background fill color.
<code>border-radius</code>	Message background border radius.
<code>bottom</code>	Offset from the bottom of the main view.
<code>color</code>	Tip text color.
<code>font-size</code>	Font size.
<code>font-family</code>	Font family.
<code>opacity</code>	Message background opacity.
<code>padding</code>	Padding around the message text.

The tip message can be localized. See [Localization of user interface elements](#) for more information.

Example – To set up semi-transparent tip message with white Arial 12px font, 50 pixels offset from the bottom of the main view, padding, and a rounded border:

```
.s7mixedmediaviewer .s7flyoutzoomview .s7tip {
bottom: 50px;
color: #ffffff;
font-family: Arial;
font-size: 12px;
padding-bottom: 10px;
padding-top: 10px;
padding-left: 12px;
padding-right: 12px;
background-color: #000000;
border-radius: 4px;
opacity: 0.5;
filter: alpha(opacity = 50);
}
```

Focus highlight

Input focus highlight displayed around focused viewer UI element is controlled with the CSS class selector.

CSS properties

The appearance is controlled with the following CSS class selector:

```
.s7mixedmediaviewer *:focus
```

CSS property	Description
outline	Focus highlight style.

Example – to disable the default browser focus highlight for all viewer user interface elements, add the following CSS selector to viewer's style sheet:

```
.s7mixedmediaviewer *:focus {
  outline: none;
}
```

Full screen button

This button causes the viewer to enter or exit full screen mode when clicked by the user. It is used when the viewer is displaying images or spin sets. This button is not displayed if the viewer works in pop-up mode and the system does not support native full screen. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7fullscreenbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the "full screen" state and *selected='false'* corresponds to the "normal" state.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a full screen button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states when selected or not selected:

```
.s7mixedmediaviewer .s7fullscreenbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='false'][state='up'] {
background-image:url(images/enterFullBtn_up.png);
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='false'][state='over'] {
background-image:url(images/enterFullBtn_over.png);
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='false'][state='down'] {
background-image:url(images/enterFullBtn_down.png);
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='false'][state='disabled'] {
background-image:url(images/enterFullBtn_disabled.png);
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='true'][state='up'] {
background-image:url(images/exitFullBtn_up.png);
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='true'][state='over'] {
background-image:url(images/exitFullBtn_over.png);
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='true'][state='down'] {
background-image:url(images/exitFullBtn_down.png);
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='true'][state='disabled'] {
background-image:url(images/exitFullBtn_disabled.png);
}
```

Main swatches

Main Swatches consist of a row of thumbnail images with optional scroll buttons on the left and right side. Scroll buttons are only visible on the desktop if all thumbnails cannot fit into the width of the container. On mobile devices, or if thumbnails can fit into the container width, scroll buttons are not shown.

The appearance of the swatches container is controlled with the CSS class selector:

```
.s7mixedmediaviewer .s7swatches
```

CSS properties of the swatches

height	The height of the swatches.
bottom	The vertical swatches offset relative to the viewer container.

Example – to set up swatches with a height of 100 pixels.

```
.s7mixedmediaviewer .s7swatches {
height: 100px;
}
```

The spacing between the swatch thumbnails is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7swatches .s7thumbcell
```


CSS property	Description
margin	The size of horizontal and vertical margin around each thumbnail. Actual thumbnail spacing equals to the sum of left and right margin set for <code>.s7thumbcell</code> .

Example

To set spacing to ten pixels both vertically and horizontally.

```
.s7mixedmediaviewer .s7swatches .s7thumbcell {
  margin: 5px;
}
```

The appearance of the individual thumbnail is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7swatches .s7thumb
```

CSS property	Description
width	Width of the thumbnail.
height	Height of the thumbnail.
border	Border of the thumbnail.



Note: Thumbnail supports the `state` attribute selector, which can be used to apply different skins to different thumbnail states. In particular, `state="selected"` corresponds to the thumbnail for the image that is currently displayed in the main view, `state="default"` corresponds to the rest of thumbnails, and `state="over"` is used on mouse hover.

Example – to set up thumbnails that are 56 x 56 pixels, have a light grey default border, and a dark grey selected border.

```
.s7mixedmediaviewer .s7swatches .s7thumb {
  width: 56px;
  height: 56px;
}
.s7mixedmediaviewer .s7swatches .s7thumb[state="default"] {
  border: 1px solid #dddddd;
}
.s7mixedviewer .s7swatches .s7thumb[state="selected"] {
  border: 1px solid #666666;
}
```

The type of the asset is displayed as an icon overlaid on top of thumbnail image and is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7swatches .s7thumb .s7thumboverlay
```

CSS property	Description
width	Width of the icon overlay.

CSS property	Description
height	Height of the icon overlay.

The overlay supports the `type` attribute selector with the following possible values: `image` (for single images), `swatchset` (for swatch sets), `spinset` (for spin sets) and `video` (for single videos or adaptive video sets).

Example – to set up icon overlays for spin sets, swatch sets, and videos:

```
.s7mixedmediaviewer .s7swatches .s7thumb .s7thumboverlay[type="swatchset"] {
  background-image: url(images/v2/ThumbOverlaySwatchSet.png);
}
.s7mixedmediaviewer .s7swatches .s7thumb .s7thumboverlay[type="spinset"] {
  background-image: url(images/v2/ThumbOverlaySpinSet.png);
}
.s7mixedmediaviewer .s7swatches .s7thumb .s7thumboverlay[type="video"] {
  background-image: url(images/v2/ThumbOverlayVideo.png);
}
```

The appearance of the left and right scroll buttons are controlled with the following CSS class selectors:

```
.s7mixedmediaviewer .s7swatches .s7scrollleftbutton
.s7mixedmediaviewer .s7swatches .s7scrollrightbutton
```

It is not possible to position scroll buttons using CSS `top`, `left`, `bottom`, and `right` properties. Instead, the viewer logic positions them automatically.

CSS property	Description
width	Width of the scroll button.
height	Height of the scroll button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states: `up`, `down`, `over`, and `disabled`.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up scroll buttons that are 56 x 56 pixels and have different artwork for each state.

```
.s7mixedmediaviewer .s7swatches .s7scrollleftbutton {
  background-size: auto;
  width: 56px;
  height: 56px;
}
.s7mixedmediaviewer .s7swatches .s7scrollleftbutton[state="up"] {
  background-image: url(images/v2/ScrollLeftButton_up.png);
}
.s7mixedmediaviewer .s7swatches .s7scrollleftbutton[state="over"] {
```

```
background-image:url(images/v2/ScrollLeftButton_over.png);
}
.s7mixedmediaviewer .s7swatches .s7scrollleftbutton[state="down"]{
background-image:url(images/v2/ScrollLeftButton_down.png);
}
.s7mixedmediaviewer .s7swatches .s7scrollleftbutton[state="disabled"]{
background-image:url(images/v2/ScrollLeftButton_disabled.png);
}
.s7mixedmediaviewer .s7swatches .s7scrollrightbutton {
background-size: auto;
width: 56px;
height: 56px;
}
.s7mixedmediaviewer .s7swatches .s7scrollrightbutton[state="up"]{
background-image:url(images/v2/ScrollRightButton_up.png);
}
.s7mixedmediaviewer .s7swatches .s7scrollrightbutton[state="over"]{
background-image:url(images/v2/ScrollRightButton_over.png);
}
.s7mixedmediaviewer .s7swatches .s7scrollrightbutton[state="down"]{
background-image:url(images/v2/ScrollRightButton_down.png);
}
.s7mixedmediaviewer .s7swatches .s7scrollrightbutton[state="disabled"]{
background-image:url(images/v2/ScrollRightButton_disabled.png);
}
```

Main viewer area

The main view area is the area occupied by the main view and swatches. It is usually set to fit the available device screen when no size is specified.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7mixedmediaviewer
```

CSS property	Description
width	The width of the viewer.
height	The height of the viewer.
background-color	Background color in hexadecimal format.

Example – to set up a viewer with a white background (#FFFFFF) and make its size 512 x 288 pixels.

```
.s7mixedmediaviewer {
background-color: #FFFFFF;
width: 512px;
height: 288px;
}
```

Mutable volume

The mutable volume control initially appears as a button that lets a user mute or unmute the video player sound.

When a user rolls over the button, a slider appears that allows a user to set the volume. The mutable volume control can be sized, skinned, and positioned, relative to the control bar that contains it, by CSS.

The appearance of the mutable volume area is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7mutablevolume
```

CSS properties of the mutable volume

top	Position from the top border, including padding.
right	Position from the right border, including padding.
width	The width of the mutable volume control.
height	The height of the mutable volume control.
background-color	The color of the mutable volume control.

The mute/unmute button appearance is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7mutablevolume .s7mutebutton
```

You can control background image for each button state. The size of the button is inherited from the size of the volume control.

CSS properties of the button image

background-image	The image displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports both the `state` and `selected` attribute selectors, which can be used to apply different skins to different button states. In particular, `selected='true'` corresponds to the "muted" state and `selected='false'` corresponds to the "unmuted" state.

The vertical volume bar area is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7mutablevolume .s7verticalvolume
```

CSS properties of the vertical volume bar area

background-color	The background color of the vertical volume.
width	The width of the vertical volume.
height	The height of the vertical volume.

The track inside vertical volume control is controlled with the following CSS class selectors:

```
.s7mixedmediaviewer .s7mutablevolume .s7verticalvolume .s7track  
.s7mixedmediaviewer .s7mutablevolume .s7verticalvolume .s7filledtrack
```

CSS properties of the vertical volume control

background-color	The background color of the vertical volume control.
width	Width of the vertical volume control.
height	Height of the vertical volume control.

The vertical volume knob is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7mutablevolume .s7verticalvolume .s7knob
```

CSS properties of the vertical volume control knob

background-image	Vertical volume control knob artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
width	Width of the vertical volume control knob.
height	Height of the vertical volume control knob.
left	Horizontal position of the vertical volume control knob.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Examples

To set up a mute button that is 32 x 32 pixels and positioned 6 pixels from the top, and 38 pixels from the right edge of the control bar. Display a different image for each of the four different button states when selected or not selected.

```
.s7mixedmediaviewer .s7mutablevolume {
top:6px;
right:38px;
width:32px;
height:32px;
}
.s7mixedmediaviewer .s7mutablevolume .s7mutebutton[selected='true'][state='up'] {
background-image:url(images/mute_up.png);
}
.s7mixedmediaviewer .s7mutablevolume .s7mutebutton[selected='true'][state='over'] {
background-image:url(images/mute_over.png);
}
.s7mixedmediaviewer .s7mutablevolume .s7mutebutton[selected='true'][state='down'] {
background-image:url(images/mute_down.png);
}
.s7mixedmediaviewer .s7mutablevolume .s7mutebutton[selected='true'][state='disabled'] {
background-image:url(images/mute_disabled.png);
}
.s7mixedmediaviewer .s7mutablevolume .s7mutebutton[selected='false'][state='up'] {
background-image:url(images/unmute_up.png);
}
.s7mixedmediaviewer .s7mutablevolume .s7mutebutton[selected='false'][state='over'] {
background-image:url(images/unmute_over.png);
}
.s7mixedmediaviewer .s7mutablevolume .s7mutebutton[selected='false'][state='down'] {
background-image:url(images/unmute_down.png);
}
```

```
}
.s7mixedmediaviewer .s7mutablevolume .s7mutebutton[selected='false'][state='disabled'] {
background-image:url(images/unmute_disabled.png);
}
```

The following is an example of how you can style the volume slider within the mutable volume control.

```
.s7mixedmediaviewer .s7mutablevolume .s7verticalvolume {
width:36px;
height:83px;
left:0px;
background-color:#dddddd;
}
.s7mixedmediaviewer .s7mutablevolume .s7verticalvolume .s7track {
top:11px;
left:14px;
width:10px;
height:63px;
background-color:#666666;
}
.s7mixedmediaviewer .s7mutablevolume .s7verticalvolume .s7filledtrack {
width:10px;
background-color:#ababab;
}
.s7mixedmediaviewer .s7mutablevolume .s7verticalvolume .s7knob {
width:18px;
height:10px;
left:9px;
background-image:url(images/volumeKnob.png);
}
```

Play/Pause button

The play/pause button causes the video player to play or pause the video content when a user clicks it.

You can size, skin, and position the button, relative to the control bar that contains it, by CSS.

The following CSS class selector controls the appearance of the button:

```
.s7mixedmediaviewer .s7playpausebutton
```

CSS properties of the play/pause button

top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used.

See [CSS Sprites](#).



Note: This button supports both the *state*, *selected*, and *replay* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the "play" state and *selected='false'* corresponds to the "pause" state;

replay='true' is set when the video has reached the end and clicking on the button restarts playback from the beginning.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example

To set up a play/pause button that is 32 x 32 pixels; it is positioned six pixels from the top and left edge of the control bar, and displays a different image for each of the four different button states when selected or not selected.

```
.s7mixedmediaviewer .s7playpausebutton {
top:6px;
left:6px;
width:32px;
height:32px;
}
.s7mixedmediaviewer .s7playpausebutton[selected='true'][state='up'] {
background-image:url(images/playBtn_up.png);
}
.s7mixedmediaviewer .s7playpausebutton[selected='true'][state='over'] {
background-image:url(images/playBtn_over.png);
}
.s7mixedmediaviewer .s7playpausebutton[selected='true'][state='down'] {
background-image:url(images/playBtn_down.png);
}
.s7mixedmediaviewer .s7playpausebutton[selected='true'][state='disabled'] {
background-image:url(images/playBtn_disabled.png);
}
.s7mixedmediaviewer .s7playpausebutton[selected='false'][state='up'] {
background-image:url(images/pauseBtn_up.png);
}
.s7mixedmediaviewer .s7playpausebutton[selected='false'][state='over'] {
background-image:url(images/pauseBtn_over.png);
}
.s7mixedmediaviewer .s7playpausebutton[selected='false'][state='down'] {
background-image:url(images/pauseBtn_down.png);
}
.s7mixedmediaviewer .s7playpausebutton[selected='false'][state='disabled'] {
background-image:url(images/pauseBtn_disabled.png);
}
.s7mixedmediaviewer .s7playpausebutton[selected='true'][replay='true'][state='up'] {
background-image:url(images/replayBtn_up.png);
}
.s7mixedmediaviewer .s7playpausebutton[selected='true'][replay='true'][state='over'] {
background-image:url(images/replayBtn_over.png);
}
.s7mixedmediaviewer .s7playpausebutton[selected='true'][replay='true'][state='down'] {
background-image:url(images/replayBtn_down.png);
}
.s7mixedmediaviewer .s7playpausebutton[selected='true'][replay='true'][state='disabled'] {
background-image:url(images/replayBtn_disabled.png);
}
```

Set indicator

Set indicator is a series of dots rendered on top of main swatches when a viewer is used on a touch device. The dots help users to navigate through pages of thumbnails when scroll buttons are not available.

CSS properties of the set indicator

The appearance of the set indicator container is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7setindicator
```

CSS property	Description
background-color	The background color in hexadecimal format of the set indicator.

Example – to set up set indicator with a white background:

```
.s7mixedmediaviewer .s7setindicator {
  background-color: #FFFFFF;
}
```

The appearance of an individual set indicator dot is controlled with the CSS class selector:

```
.s7mixedmediaviewer .s7setindicator .s7dot
```

CSS property	Description
width	Width of the set indicator dot.
height	Height of the set indicator dot.
margin-left	Left margin in pixels.
margin-top	Top margin in pixels.
margin-right	Right margin in pixels.
margin-bottom	Bottom margin in pixels.
border-radius	Border radius in pixels.
background-color	Background color in hexadecimal format.



Note: Set indicator dot supports the *state* attribute selector, which can be used to apply different skins to different thumbnail states. In particular, *state="selected"* corresponds to the current page of thumbnails, *state="unselected"* corresponds to the default dot state.

Example – to set up set indicator dot to be 15 x 15 pixels, with two pixels horizontal margin, five pixels top margin, one pixel bottom margin, twelve pixels radius, #D5D3D3 default color, and #939393 active color:

```
.s7mixedmediaviewer .s7setindicator .s7dot {
  width:15px;
  height:15px;
  margin-left:2px;
  margin-top:5px;
  margin-right:2px;
  margin-bottom:1px;
}
```



```
border-radius:12px;
background-color:#D5D3D3;
}
.s7mixedmediaviewer .s7setindicator .s7dot[state="selected"] {
background-color:#939393;
}
```

Spin left button

Clicking or tapping this button spins the image to the left in main view. This button is not displayed on mobile phones in order to save screen real estate. Also, the button is hidden when a multi-dimensional spin set is used. You can size, skin, and position the button using CSS.

CSS properties of the spin buttons

The button is added to an internal container that is DIV controlled with the CSS class selector:

```
.s7mixedmediaviewer .s7spinbuttons
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.

The appearance of this button inside the container is controlled with the CSS class selector:

```
.s7mixedmediaviewer .s7spinbuttons .s7panleftbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.

CSS property	Description
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a spin left button that is 28 x 28 pixels, that is positioned on the left edge of inner container, and that displays a different image for each of the four different button states:

```
.s7mixedmediaviewer .s7spinbuttons .s7panleftbutton {
  position:absolute;
  left: 0px;
  width:28px;
  height:28px;
  background-size:contain;
}
.s7mixedmediaviewer .s7spinbuttons .s7panleftbutton[state='up'] {
background-image:url(images/v2/SpinLeftButton_light_up.png);
}
.s7mixedmediaviewer .s7spinbuttons .s7panleftbutton[state='over'] {
background-image:url(images/v2/SpinLeftButton_light_over.png);
}
.s7mixedmediaviewer .s7spinbuttons .s7panleftbutton[state='down'] {
background-image:url(images/v2/SpinLeftButton_light_down.png);
}
.s7mixedmediaviewer .s7spinbuttons .s7panleftbutton[state='disabled'] {
background-image:url(images/v2/SpinLeftButton_light_disabled.png);
}
```

Spin right button

Clicking or tapping this button spins the image to the right in main view. This button is not displayed on mobile phones in order to save screen real estate. Also, the button is hidden when a multi-dimensional spin set is used. You can size, skin, and position the button using CSS.

CSS properties of the spin buttons

The button is added to an internal container that is DIV controlled with the CSS class selector:

```
.s7mixedmediaviewer .s7spinbuttons
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.

CSS property	Description
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.

The appearance of this button inside the container is controlled with the CSS class selector:

```
.s7mixedmediaviewer .s7spinbuttons .s7panrightbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a spin right button that is 28 x 28 pixels, that is positioned on the right edge of inner container, and that displays a different image for each of the four different button states:

```
.s7mixedmediaviewer .s7spinbuttons .s7panrightbutton {
  position:absolute;
  right: 0px;
  width:28px;
  height:28px;
  background-size:contain;
}
.s7mixedmediaviewer .s7spinbuttons .s7panrightbutton[state='up'] {
  background-image:url(images/v2/SpinRightButton_light_up.png);
}
.s7mixedmediaviewer .s7spinbuttons .s7panrightbutton[state='over'] {
```

```
background-image:url(images/v2/SpinRightButton_light_over.png);
}
.s7mixedmediaviewer .s7spinbuttons .s7panrightbutton[state='down'] {
  background-image:url(images/v2/SpinRightButton_light_down.png);
}
.s7mixedmediaviewer .s7spinbuttons .s7panrightbutton[state='disabled'] {
  background-image:url(images/v2/SpinRightButton_light_disabled.png);
}
```

Spin view

Main view consists of the spin image when the current asset is a spin set.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7spinview
```

CSS property	Description
background-color	Background color in hexadecimal format of the spin view.

Example – to make the spin view transparent.

```
.s7mixedmediaviewer .s7spinview {
  background-color: transparent;
}
```

Spin view icon effect

The spin indicator is overlaid on the spin view area. It is displayed when the image is in a reset state and it also depends on the `iconeffect` parameter.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7spinview .s7iconeffect
```

CSS property	Description
background-image	Spin indicator artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
width	Spin indicator width.
height	Spin indicator height.

Spin indicator supports the `state` attribute selector which is set to `spin_1D` in case of single-dimensional spin set and to `spin_2D` in case of multi-dimensional spin set.

Example – to set up a 100 x 100 pixels zoom indicator.

```
.s7mixedmediaviewer .s7spinview .s7iconeffect {
  width: 100px;
  height: 100px;
}
.s7mixedmediaviewer .s7spinview .s7iconeffect[state="spin_1D"] {
background-image: url(images/spinIcon_1D.png);
}
.s7mixedmediaviewer .s7spinview .s7iconeffect[state="spin_2D"] {
background-image: url(images/spinIcon_2D.png);
}
```

Tooltips


On desktop systems some user interface elements like buttons have tooltips that are displayed on mouse hover.

CSS properties of the main viewer area

The appearance of tooltips is controlled with the following CSS class selector:

```
.s7tooltip
```

CSS property	Description
border-radius	Background border radius.
border-color	Background border color.
background-color	Background color.
color	Text color.
font-family	Text font name.
font-size	Text font size.

 **Note:** In case tooltip styles are customized from within the embedding web page, all properties have to contain ! IMPORTANT rule. This is not necessary if tooltips are customized in the viewer's CSS file.

Example – to set up tooltips that have a grey border with 3px corner radius, black background and white text written with Arial, 11 pixels size:

```
.s7tooltip {
  border-radius: 3px 3px 3px 3px;
  border-color: #999999;
  background-color: #000000;
  color: #FFFFFF;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 11px;
}
```

Video full screen button

The full screen button causes the viewer to enter or exit full screen mode when clicked by the user. It is used when the viewer is displaying video and is located in the control bar. This button is not displayed if the viewer works in pop-up mode and the system does not support native full screen.

You can size, skin, and position the full screen button, relative to the control bar that contains it, by CSS.

The appearance of the full screen button is controlled with the CSS class selector:

```
.s7mixedmediaviewer .s7fullscreenbutton
```

CSS properties of the full screen button

top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	The width of the full screen button.
height	The height of the full screen button.
background-image	The displayed image for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the "full screen" state and *selected='false'* corresponds to the "normal" state.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example

To set up a full screen button that is 32 x 32 pixels, and positioned 6 pixels from the top and right edge of the control bar. Also, display a different image for each of the four different button states when selected or not selected.

```
.s7mixedmediaviewer .s7fullscreenbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='false'][state='up'] {
background-image:url(images/enterFullBtn_up.png);
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='false'][state='over'] {
background-image:url(images/enterFullBtn_over.png);
```

```
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='false'][state='down'] {
background-image:url(images/enterFullBtn_down.png);
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='false'][state='disabled'] {
background-image:url(images/enterFullBtn_disabled.png);
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='true'][state='up'] {
background-image:url(images/exitFullBtn_up.png);
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='true'][state='over'] {
background-image:url(images/exitFullBtn_over.png);
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='true'][state='down'] {
background-image:url(images/exitFullBtn_down.png);
}
.s7mixedmediaviewer .s7fullscreenbutton [selected='true'][state='disabled'] {
background-image:url(images/exitFullBtn_disabled.png); }
}
```

Video player

The video player is the rectangular area where the video content is displayed within the viewer.

If the dimensions of the video that is being played does not match the dimensions of the video player, the video content is centered within the video player's rectangle display area.

The following CSS class selector controls the appearance of the video player:

```
.s7mixedmediaviewer .s7videoplayer
```

CSS properties of the video player

background-color	The background color of the video player.
------------------	---

The error message that is displayed if the system is unable to play the video can be localized. See [Localization of user interface elements](#) for more information.

Example – To make the video player transparent:

```
.s7mixedmediaviewer .s7videoplayer {
background-color: transparent;
}
```

Captions are put into internal container inside the video player. The position of that container is controlled by supported WebVTT positioning operators. The caption text itself is inside that container; its style is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7videoplayer .s7caption
```

CSS properties of captions

CSS property	Description
background-color	Caption text background.
color	Caption text color.
font-weight	Font weight.

CSS property	Description
font-size	Font size.
font-family	Font family.

Example – To set up caption text to be 14 pixel light gray Arial on a semi-transparent black background:

```
.s7mixedmediaviewer .s7videoplayer .s7caption {
  background-color: rgba(0,0,0,0.75);
  color: #e6e6e6;
  font-weight: normal;
  font-size: 14px;
  font-family: Arial,Helvetica,sans-serif;
}
```

The appearance of the buffering animation is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7videoplayer .s7waiticon
```

CSS properties of wait icon

CSS property	Description
width	Animation icon width.
height	Animation icon height.
margin-left	Animation icon left margin, normally minus half of the icon's width.
margin-top	Animation icon top margin, normally minus half of the icon's height.
background-image	Knob artwork.

Example – To set up a buffering animation to be 101 pixels wide, 29 pixels high:

```
.s7mixedmediaviewer .s7videoplayer .s7waiticon {
  width: 101px;
  height: 29px;
  margin-left: -50px;
  margin-top: -15px;
  background-image: url(images/sdk/busyicon.gif);
}
```

Video player icon effect

The play icon is overlaid on the video view area. It displays when the video is paused, or when the end of the video is reached, and it also depends on the `iconeffect` parameter.

The appearance of the play icon is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7videoplayer .s7iconeffect
```


CSS properties of the play icon

background-image	The displayed image for the play icon.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
width	The width of the play icon.
height	The height of the play icon.

Icon effect supports the `state` attribute selector. `state="play"` is used when the video is paused in the middle of playback, and `state="replay"` is used when the play head is in the end of the stream.

Example

Setup a 100 x 100 pixel play icon.

```
.s7mixedmediaviewer .s7videoplayer .s7iconeffect {
  width: 100px;
  height: 100px;}
.s7mixedmediaviewer .s7videoplayer .s7iconeffect[state="play"] {
  background-image: url(images/playIcon.png);
}
.s7mixedmediaviewer .s7videoplayer .s7iconeffect[state="replay"] {
  background-image: url(images/replayIcon.png);
}
```

Video scrubber

The video scrubber is the horizontal slider control that lets a user dynamically seek to any time position within the currently playing video.

The scrubber 'knob' also moves as the video plays to indicate the current time position of the video during playback. The video scrubber always takes the whole width of the control bar. It is possible to skin the video scrubber, change its height and vertical position, by CSS.

The general appearance of the video scrubber is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7videoscrubber
.s7mixedmediaviewer .s7videoscrubber .s7videotime
.s7mixedmediaviewer .s7videoscrubber .s7knob
```

CSS properties of the video scrubber

top	Position from the top border, including padding.
bottom	Position from the bottom border, including padding.
height	Height of the video scrubber.
background-color	The color of the video scrubber.

The following CSS class selectors track background, play, and load indicators:

```
.s7mixedmediaviewer .s7videoscubber .s7track
.s7mixedmediaviewer .s7videoscubber .s7trackloaded
.s7mixedmediaviewer .s7videoscubber .s7trackplayed
```

CSS properties of the track

height	Height of the corresponding track.
background-color	The color of the corresponding track.

The following CSS class selector controls the knob:

```
.s7mixedmediaviewer .s7videoscubber .s7knob
```

CSS properties of the knob

top	Vertical knob offset.
width	Width of knob.
height	Height of knob.
background-image	Knob artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

The following CSS class selector controls the time played bubble:

```
.s7mixedmediaviewer .s7videoscubber .s7videotime
```

CSS properties of the time played bubble

font-family	The font family to use for the time display text.
font-size	The font size to use for the time display text.
color	The font color to use for the time display text.
width	Bubble area width.
height	Bubble area height.
padding	Bubble area padding.
background-image	Bubble artwork.

background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
text-align	Alignment of text with the bubble area.

The video scrubber tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example

To set up a mixed media viewer with a video scrubber with custom track colors that is 10 pixels tall, and positioned 10 pixels and 35 pixels from the top and left edges of the control bar.

```
.s7mixedmediaviewer .s7videoscrubber {
top:10px;
left:35px;
height:10px;
background-color:#AAAAAA;
}
.s7mixedmediaviewer .s7videoscrubber .s7track {
height:10px;
background-color:#444444;
}
.s7mixedmediaviewer .s7videoscrubber .s7trackloaded {
height:10px;
background-color:#666666;
}
.s7mixedmediaviewer .s7videoscrubber .s7trackplayed {
height:10px;
background-color:#888888;
}
```

Video time

The video time is the numeric display that shows the current time and duration of the currently playing video.

The video time font family, font size, and font color are among the properties that CSS can control. It can also be positioned, relative to the control bar that contains it, by CSS.

The appearance of the video time is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7videotime
```

CSS properties of video time

top	Position from the top border, including padding.
right	Position from the right border, including padding.
width	The width of video time control. This property is required for Internet Explorer 8 or greater to function properly.
font-family	The font family to use for the time display text.
font-size	The font size to use for the time display text.

color	The font color to use for the time display text.
-------	--

Example

Set the video time to light gray (hexadecimal #BBBBBB), sized at 12 pixels, positioned 15 pixels from the top of the control bar, and 80 pixels from the right edges of the control bar.

```
.s7mixedmediaviewer .s7videotime {
top:15px;
right:80px;
font-size:12px;
color:#BBBBBB;
width:60px;
}
```

Zoom in button

Clicking or tapping this button zooms in on an image in the main view. This button does not display on mobile phones in order to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7zoominbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom in button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7mixedmediaviewer .s7zoominbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7mixedmediaviewer .s7zoominbutton [state='up'] {
background-image:url(images/v2/ZoomInButton_dark_up.png);
}
.s7mixedmediaviewer .s7zoominbutton [state='over'] {
background-image:url(images/v2/ZoomInButton_dark_over.png);
}
.s7mixedmediaviewer .s7zoominbutton [state='down'] {
background-image:url(images/v2/ZoomInButton_dark_down.png);
}
.s7mixedmediaviewer .s7zoominbutton [state='disabled'] {
background-image:url(images/v2/ZoomInButton_dark_disabled.png);
}
```

Zoom out button

Clicking or tapping this button zooms out on an image in the main view. This button does not display on mobile phones in order to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7zoomoutbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom out button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7mixedmediaviewer .s7zoomoutbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7mixedmediaviewer .s7zoomoutbutton [state='up'] {
background-image:url(images/v2/ZoomOutButton_dark_up.png);
}
.s7mixedmediaviewer .s7zoomoutbutton [state='over'] {
background-image:url(images/v2/ZoomOutButton_dark_over.png);
}
.s7mixedmediaviewer .s7zoomoutbutton [state='down'] {
background-image:url(images/v2/ZoomOutButton_dark_down.png);
}
.s7mixedmediaviewer .s7zoomoutbutton [state='disabled'] {
background-image:url(images/v2/ZoomOutButton_dark_disabled.png);
}
```

Zoom reset button


Clicking or tapping this button resets an image in the main view. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7zoomresetbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

 **Note:** This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom reset button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7mixedmediaviewer .s7zoomresetbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7mixedmediaviewer .s7zoomresetbutton [state='up'] {
background-image:url(images/v2/ZoomResetButton_dark_up.png);
}
.s7mixedmediaviewer .s7zoomresetbutton [state='over'] {
background-image:url(images/v2/ZoomResettButton_dark_over.png);
}
.s7mixedmediaviewer .s7zoomresetbutton [state='down'] {
background-image:url(images/v2/ZoomResetButton_dark_down.png);
}
.s7mixedmediaviewer .s7zoomresetbutton [state='disabled'] {
background-image:url(images/v2/ZoomResetButton_dark_disabled.png);
}
```

Zoom view

In continuous zoom mode, main view consists of the zoomable image when the current asset is a single image..

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7zoomview
```

CSS property	Description
background-color	Background color in hexadecimal format of the main view.
cursor	Cursor displayed over the main view.

Example – to make the zoom view transparent.

```
.s7mixedmediaviewer .s7zoomview {
background-color: transparent;
}
```

On desktop systems the component supports `cursor`type attribute selector which can be applied to the `.s7zoomview` class. It controls the type of the cursor based on component state and user action. The following `cursor`type values are supported:

- default
Displayed when the image is not zoomable because of a small image resolution, or component settings, or both.
- zoomin
Displayed when the image can be zoomed in.
- reset
Displayed when the image is at maximum zoom level and can be reset to its initial state.
- drag

Displayed when the user pans the image which is in zoomed in state.

- `slide`

Displayed when the user performs image swap by doing a horizontal swipe or flick.

Zoom view icon effect

The zoom indicator is overlaid on the zoom view area. It is displayed when the image is in a reset state and it also depends on `iconeffect` parameter.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7mixedmediaviewer .s7zoomview .s7iconeffect
```

CSS property	Description
<code>background-image</code>	Zoom indicator artwork.
<code>background-position</code>	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
<code>width</code>	Zoom indicator width.
<code>height</code>	Zoom indicator height.



Note: Icon effect supports the `media-type` attribute selector, which you can use to apply different icon effects on different devices. In particular, `media-type='standard'` corresponds to desktop systems where mouse input is normally used and `media-type='multitouch'` corresponds to devices with touch input.

Example – to set up a 100 x 100 pixel zoom indicator with different art for desktop systems and touch devices.

```
.s7mixedmediaviewer .s7zoomview .s7iconeffect {
  width: 100px;
  height: 100px;
}
.s7mixedmediaviewer .s7zoomview .s7iconeffect[media-type='standard'] {
  background-image: url(images/v2/IconEffect_zoom.png);
}
.s7mixedmediaviewer .s7zoomview .s7iconeffect[media-type='multitouch'] {
  background-image: url(images/v2/IconEffect_pinch.png);
}
```

Support for Adobe Analytics tracking

The Mixed Media Viewer supports Adobe Analytics tracking out-of-the-box.

Out-of-the-box tracking

The Mixed Media Viewer supports Adobe Analytics tracking out-of-the-box. To enable tracking, pass the proper company preset name as `config2` parameter.

The viewer also sends a single tracking HTTP request to the configured Image Server with the viewer type and version information.

Custom tracking

To integrate with third-party analytics systems it is necessary to listen to the `trackEvent` viewer callback and process the `eventInfo` argument of the callback function as necessary. The following code is an example of such handler function:

```
var mixedMediaViewer = new s7viewers.MixedMediaViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/Mixed_Media_Set_Sample",
    "serverurl": "http://s7dl.scene7.com/is/image/",
    "videoseverurl": "http://s7dl.scene7.com/is/content/"
  },
  "handlers": {
    "trackEvent": function(objID, compClass, instName, timeStamp, eventInfo) {
      //identify event type
      var eventType = eventInfo.split(",")[0];
      switch (eventType) {
        case "LOAD":
          //custom event processing code
          break;
        //additional cases for other events
      }
    }
  }
});
```

The viewer tracks the following SDK user events:

SDK user event	Sent when...
LOAD	viewer is loaded first.
SWAP	an asset is swapped in the viewer using <code>setAsset()</code> API.
ZOOM	an image is zoomed.
PAN	an image is panned.
SWATCH	an image is changed by clicking or tapping on a swatch.
PLAY	playback is started.
PAUSE	playback is paused.
STOP	playback is stopped.
MILESTONE	playback reaches one of the following millstones: 0%, 25%, 50%, 75%, and 100%.
SPIN	spin is performed.

HTTPS video delivery



Note: Secure Video Delivery only applies to AEM 6.2 with the installation of [Feature Pack-13480](#) and to AEM 6.1 with installation of [Feature Pack NPR-15011](#).

Provided that the viewer works in configuration as outlined at the beginning of this section, published video delivery can happen both in HTTPS (secure) and HTTP (insecure) modes. In a default configuration, the video delivery protocol strictly follows the delivery protocol of the embedding web page. However, it is possible to force HTTPS video delivery without regard to the protocol used by embedding the web page using the [VideoPlayer.ssl](#) configuration attribute. (Note that video preview in Author mode is always delivered securely over HTTPS.)

Depending on the method of publishing Dynamic Media video that you use in AEM, the `VideoPlayer.ssl` configuration attribute is applied differently as demonstrated in the following:

- If you publish a Dynamic Media video with a URL, you append `VideoPlayer.ssl` to the URL. For example, to force secure video delivery, you append `&VideoPlayer.ssl=on` to the end of the following viewer URL example:

~~`https://demos-pub.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/MixedMediaViewer.js?asset=/content/dam/Geometrix-Outdoors-New-Launch/backpack/backpack_mixed_media`~~

See also [\(AEM 6.2\) Linking URLs to your Web Application](#) or [\(AEM 6.1\) Linking URLs to your Web Application](#)

- If you publish a Dynamic Media video with embed code, you add `VideoPlayer.ssl` to the list of other viewer configuration parameters in the embed code snippet. For example, to force HTTPS video delivery, you append `&VideoPlayer.ssl=on` as in the following example:

```
<style type="text/css">
  #s7mixedmedia_div.s7mixedmediaviewer{
    width:100%;
    height:auto;
  }
</style>
<script type="text/javascript"
src="https://demos-pub.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/MixedMediaViewer.js"></script>
<div id="s7mixedmedia_div"></div>
<script type="text/javascript">
  var s7mixedmediaviewer = new s7viewers.MixedMediaViewer({
    "containerId" : "s7mixedmedia_div",
    "params" : {
      "VideoPlayer.ssl" : "on",
      "serverurl" : "https://adobedemo62-h.assetsadobe.com/is/image",
      "contenturl" : "https://demos-pub.assetsadobe.com/",
      "config" : "/etc/dam/presets/viewer/MixedMedia_light",
      "config2": "/etc/dam/presets/analytics",
      "videoserverurl": "https://gateway-na.assetsadobe.com/DMGateway/public/demoCo",
      "asset" : "/content/dam/Geometrix-Outdoors-New-Launch/backpack/backpack_mixed_media" }
    }).init();
</script>
```

See also [\(AEM 6.2\) Embedding the Video on a Web Page](#) or [\(AEM 6.1\) Embedding the Video on a Web Page](#).

Localization of user interface elements

Certain content that the Mixed Media Viewer displays is subject to localization. This includes zoom buttons, spin buttons, video controls, close button full screen button and swatch scroll buttons.

Every textual content in the viewer that can be localized is represented by a special Viewer SDK identifier called SYMBOL. Any SYMBOL has a default associated text value for the English locale ("en") supplied with the out-of-the-box viewer. It may also have user-defined values set for as many locales as needed.

When the viewer starts, it checks the current locale to see if there is a user-defined value for each supported SYMBOL for the locale. If there is, it uses the user-defined value; otherwise, it falls back to the out-of-the-box default text.

User-defined localization data can be passed to the viewer as a localization JSON object. Such object contains the list of supported locales, SYMBOL text values for each locale, and the default locale.

An example of such a localization object is the following:

```
{
  "en": {
    "CloseButton.TOOLTIP": "Close",
    "ZoomInButton.TOOLTIP": "Zoom In"
  },
  "fr": {
    "CloseButton.TOOLTIP": "Fermer",
    "ZoomInButton.TOOLTIP": "Agrandir"
  },
  defaultLocale: "en"
}
```

In the above example, the localization object defines two locales ("en" and "fr") and provides localization for two user interface elements in each locale.

The web page code should pass the localization object to the viewer constructor as a value of the `localizedTexts` field of the configuration object. An alternative option is to pass the localization object by calling the `setLocalizedTexts(localizationInfo)` method.

The following SYMBOLs are supported:

SYMBOL	Tool tip for...
CloseButton.TOOLTIP	Close button.
ZoomInButton.TOOLTIP	Zoom in button.
ZoomOutButton.TOOLTIP	Zoom out button.
ZoomResetButton.TOOLTIP	Zoom reset button.
FlyoutZoomView.TIP_BUBBLE_OVER	Desktop systems in inline zoom mode.
FlyoutZoomView.TIP_BUBBLE_TAP	Touch devices in inline zoom mode.
FullScreenButton.TOOLTIP_SELECTED	Full screen button in normal state.
FullScreenButton.TOOLTIP_UNSELECTED	Full screen button in full screen state.
ClosedCaptionButton.TOOLTIP_SELECTED	Selected close caption button state.
ClosedCaptionButton.TOOLTIP_UNSELECTED	Unselected closed caption button state.
ScrollLeftButton.TOOLTIP	Scroll left button.

SYMBOL	Tool tip for...
<code>ScrollRightButton.TOOLTIP</code>	Scroll right button.
<code>ScrollUpButton.TOOLTIP</code>	Scroll up button.
<code>ScrollDownButton.TOOLTIP</code>	Scroll down button.
<code>PanLeftButton.TOOLTIP</code>	Spin left button.
<code>PanRightButton.TOOLTIP</code>	Spin right button.
<code>PlayPauseButton.TOOLTIP_SELECTED</code>	Selected play pause button state.
<code>PlayPauseButton.TOOLTIP_UNSELECTED</code>	Deselected play pause button state.
<code>PlayPauseButton.TOOLTIP_REPLAY</code>	Play pause button state.
<code>VideoScrubber.TOOLTIP</code>	Video scrubber.
<code>VideoTime.TOOLTIP</code>	Video time on control bar.
<code>MutableVolume.TOOLTIP_SELECTED</code>	Selected mutable volume state.
<code>MutableVolume.TOOLTIP_UNSELECTED</code>	Deselected mutable volume.
<code>VideoPlayer.ERROR</code>	Error message that appears when no video playback is possible.

Full Screen Support

The viewer supports full screen operation mode.

On modern desktop browsers, except Internet Explorer 10 and older, and on some touch devices, the viewer uses "native" full screen mode. This mode means that the entire device screen is occupied by the viewer content.

On iOS devices and on older Internet Explorer browsers, the viewer uses "simulated" full screen mode instead. In this mode, the viewer simply resizes to take the full area of the web browser window. Also, the web browser Chrome and other windows are still visible on the screen.

An end user enters and leaves full screen mode by pressing the Full Screen button in the viewer user interface. When "native" full screen mode is used on desktop, it is also possible to exit it by pressing **Esc**.

Viewer SDK namespace

The viewer is built of many Viewer SDK components. In most cases, the web page does not need to interact with SDK components API directly; all common needs are covered in the viewer API itself.

However, some advanced use cases require that the web page obtain a reference to an inner SDK component using the `getComponent()` viewer API and then use all the flexibility of the APIs of SDK itself.

The namespace that is used to load and initialize SDK components by the viewer depends on the environment in which the viewer is operating. If the viewer is running in AEM (Adobe Experience Manager), the viewer loads SDK components into `s7viewers.s7sdk` namespace. Likewise, the viewer served from Scene7 Publishing System loads the SDK into `s7classic.s7sdk`.

In either case, the namespace used by the SDK inside the viewer has either `s7viewers` or `s7classic` as the prefix. And, it is different from the plain `s7sdk` namespace used in the SDK User Guide or SDK API documentation. For that reason, it is important to use a fully qualified SDK namespace when you write custom application code that communicates with internal viewer components.

For example, if you plan to listen to `StatusEvent.NOTF_VIEW_READY` event and the viewer is served from AEM, the fully qualified event type is `s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY`, and the event listener code looks similar to the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var zoomView = <instance>.getComponent("zoomView");
    zoomView.addEventListener(s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});
```

The same code for the viewer served from Scene7 Publishing System looks like the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var zoomView = <instance>.getComponent("zoomView");
    zoomView.addEventListener(s7classic.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});
```

Spin

Spin Viewer is an image viewer that provides a 360-degree view of the image or even multi-dimensional view if appropriate spin set is being used. It has zoom and spin tools, full screen support, and an optional close button. It is designed to work on desktops and mobile devices.



Note: Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.

Viewer type 503.

See [System requirements](#).

Demo URL

https://s7d9.scene7.com/s7viewers/html5/SpinViewer.html?asset=Scene7SharedAssets/SpinSet_Sample&stagesize=500,400

Using Spin Viewer

Spin Viewer represents a main JavaScript file and a set of helper files (a single JavaScript include with all Viewer SDK components used by this particular viewer, assets, CSS) downloaded by the viewer in runtime.

Spin Viewer can be used both in pop-up mode using production-ready HTML page provided with IS-Viewers or in embedded mode, where it is integrated into target web page using documented API.

Configuration and skinning are similar to that of the other viewers. All skinning can be achieved via custom CSS.

See [Command reference common to all viewers – Configuration attributes](#) and [Command reference common to all Viewers – URL](#)

Interacting with Spin Viewer

Spin Viewer supports the following touch gestures that are common in other mobile applications. When the viewer cannot process a user's swipe gesture it forwards the event to the web browser to perform a native page scroll. This allows the user to navigate through the page even if the viewer occupies most of the device screen area.

Gesture	Description
Double tap	Zooms in one level until maximum magnification is reached. The next double tap gesture resets the viewer to the initial viewing state.
Pinch	Zooms in or out on the image.
Horizontal swipe or flick	If the image is in a reset state it spins through the set horizontally. If the image is zoomed in, it moves the image horizontally. If image is moved to the view edge and a swipe is still done in that direction, the gesture performs a native page scroll.
Vertical swipe or flick	If the image is in a reset state it changes the vertical view angle in case a multi-dimensional spin set is used. In a one-dimensional spin set, or when a multi-dimensional spin set is on the last or the first axis, so that the vertical swipe does not result in a vertical view angle change, the gesture performs a native page scroll. If the image is zoomed in, it moves the image vertically. If image is moved to the view edge and a swipe is still done in that direction, the gesture performs a native page scroll.



Note: The viewer also supports both touch input and mouse input on Windows devices with touchscreen and mouse. This support, however, is limited to Chrome, Internet Explorer 11, and Edge web browsers only.

This viewer is fully keyboard accessible.

See [Keyboard accessibility and navigation](#).

Embedding Spin Viewer

Different web pages have different needs for viewer behavior. Sometimes a web page provides a link that, when clicked, opens the viewer in a separate browser window. In other cases, it is necessary to embed the viewer right in the hosting page. In the latter case, the web page may have a static page layout, or use responsive design that displays differently on different devices or for different browser window sizes. To accommodate these needs, the viewer supports three primary operation modes: pop-up, fixed size embedding, and responsive design embedding.

About pop-up mode

In pop-up mode, the viewer is opened in a separate web browser window or tab. It takes the entire browser window area and adjusts in case the browser is resized, or a mobile device's orientation is changed.

Pop-up mode is the most common for mobile devices. The web page loads the viewer using `window.open()` JavaScript call, properly configured A HTML element, or any other suitable method.

It is recommended that you use an out-of-the-box HTML page for pop-up operation mode. In this case, it is called `SpinViewer.html` and is located within the `html5/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/SpinViewer.html
```

You can achieve visual customization by applying custom CSS.

The following is an example of HTML code that opens the viewer in a new window:

```
<a href="http://s7d1.scene7.com/s7viewers/html5/SpinViewer.html?asset=Scene7SharedAssets/SpinSet_Sample&stagesize=500,400" target="_blank">Open popup viewer</a>
```

About fixed size embedding mode and responsive design embedding mode

In the embedded mode, the viewer is added to the existing web page, which may already have some customer content not related to the viewer. The viewer normally occupies only a part of a web page's real estate.

The primary use cases are web pages oriented for desktops or tablet devices, and also responsive design pages that adjust layout automatically depending on the device type.

Fixed size embedding is used when the viewer does not change its size after initial load. This is the best choice for web pages that have a static layout.

Responsive design embedding assumes that the viewer may need to resize at runtime in response to the size change of its container `DIV`. The most common use case is adding a viewer to a web page that uses a flexible page layout.

In responsive design embedding mode, the viewer behaves differently depending on the way web page sizes its container `DIV`. If the web page sets only the width of the container `DIV`, leaving its height unrestricted, the viewer automatically chooses its height according to the aspect ratio of the asset that is used. This functionality ensures that the asset fits perfectly into the view without any padding on the sides. This use case is the most common for web pages using responsive design layout frameworks like Bootstrap, Foundation, and so on.

Otherwise, if the web page sets both the width and the height for the viewer's container `DIV`, the viewer fills just that area and follows the size that the web page layout provides. A good example may be embedding the viewer into a modal overlay, where the overlay is sized according to web browser window size.

Fixed size embedding

You add the Spin Viewer to a web page by doing the following:

1. Adding the viewer JavaScript file to your web page.
 2. Defining the container `DIV`.
 3. Setting the viewer size.
 4. Creating and initializing the viewer.
1. Adding the viewer JavaScript file to your web page.

Creating a viewer requires that you add a script tag in the HTML head. Before you can use viewer API, be sure that you include `SpinViewer.js`. `SpinViewer.js` is located under the `html5/js/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/js/SpinViewer.js
```

You can use a relative path if the viewer is deployed on one of the Adobe Scene7 servers and it is served from the same domain. Otherwise, you specify a full path to one of Adobe Scene7 servers that have the IS-Viewers installed.

The relative path looks like the following:

```
<script language="javascript" type="text/javascript"
src="/s7viewers/html5/js/SpinViewer.js"></script>
```



Note: You should only reference the main viewer JavaScript include file on your page. You should not reference any additional JavaScript files in the web page code which might be downloaded by the viewer's logic in runtime. In particular, do not directly reference `HTML5 SDK Utils.js` library loaded by the viewer from `/s7viewers` context path (so-called consolidated SDK include). The reason is that the location of `Utils.js` or similar runtime viewer libraries is fully managed by the viewer's logic and the location changes between viewer releases. Adobe does not keep older versions of secondary viewer includes on the server.

As a result, putting a direct reference to any secondary JavaScript include used by the viewer on the page breaks the viewer functionality in the future when a new product version is deployed.

2. Defining the container DIV.

Add an empty DIV element to the page where you want the viewer to appear. The DIV element must have its ID defined because this ID is passed later to the viewer API.

The placeholder DIV is a positioned element, meaning that the `position` CSS property is set to `relative` or `absolute`.

The following is an example of a defined placeholder DIV element:

```
<div id="s7viewer" style="position:relative"></div>
```

3. Setting the viewer size

You can set the static size for the viewer by either declaring it for `.s7spinviewer` top-level CSS class in absolute units, or by using `stagesize` modifier.

You can put sizing in CSS directly on the HTML page, or in a custom viewer CSS file, which is then later assigned to a viewer preset record in Scene7 Publishing System, or passed explicitly using a `style` command.

See [Customizing Spin Viewer](#) for more information about styling the viewer with CSS.

The following is an example of defining a static viewer size in HTML page:

```
#s7viewer.s7spinviewer {
width: 640px;
height: 480px;
}
```

You can set the `stagesize` modifier either in the viewer preset record in Scene7 Publishing System, or pass it explicitly with the viewer initialization code with `params` collection, or as an API call as described in the Command Reference section, like the following:

```
spinViewer.setParam("stagesize",
"640,480");
```

A CSS-based approach is recommended and is used in this example.

4. Creating and initializing the viewer.

When you have completed the steps above, you create an instance of `s7viewers.SpinViewer` class, pass all configuration information to its constructor and call `init()` method on a viewer instance. Configuration information is passed to the constructor as a JSON object. At minimum, this object has `containerId` field that holds the name of viewer container ID and nested `params` JSON object with configuration parameters that the viewer supports. In this case of `params` object, it must have at least the Image Serving URL passed as `serverUrl` property and initial asset as `asset` parameter. JSON-based initialization API lets you create and start the viewer with single line of code.

It is important to have the viewer container added to the DOM so that the viewer code can find the container element by its ID. Some browsers delay building DOM until the end of the web page. For maximum compatibility, call the `init()` method just before the closing `BODY` tag, or on the `body onload()` event.

At the same time, the container element should not necessarily be a part of the web page layout just yet. For example, it may be hidden using the `display:none` style assigned to it. In this case, the viewer delays its initialization process until the moment when the web page brings the container element back to the layout. When this action occurs, the viewer load automatically resumes.

The following is an example of creating a viewer instance, passing the minimum necessary configuration options to the constructor and calling the `init()` method. The example assumes `spinViewer` is the viewer instance, `s7viewer` is the name of placeholder `DIV`, `http://s7d1.scene7.com/is/image/` is the Image Serving URL, and `Scene7SharedAssets/SpinSet_Sample` is the asset.

```
<script type="text/javascript">
var spinViewer = new s7viewers.SpinViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/SpinSet_Sample",
    "serverurl": "http://s7d1.scene7.com/is/image/"
  }
}).init();
</script>
```

The following code is a complete example of a trivial web page that embeds the Spin Viewer with a fixed size:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7d1.scene7.com/s7viewers/html5/js/SpinViewer.js"></script>
<style type="text/css">
#s7viewer.s7spinviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var spinViewer = new s7viewers.SpinViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/SpinSet_Sample",
    "serverurl": "http://s7d1.scene7.com/is/image/"
  }
}).init();
</script>
</body>
</html>
```

Responsive design embedding with unrestricted height

With responsive design embedding, the web page normally has some kind of flexible layout in place that dictates the runtime size of the viewer's container `DIV`. For purposes of this example, assume that the web page allows the viewer's container `DIV` to take 40% of the web browser window size, leaving its height unrestricted. The resulting web page HTML code looks like the following:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
.holder {
  width: 40%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>
```

Adding the viewer to such a page is similar to fixed size embedding, with the only difference being that you do not need to explicitly define viewer size.

1. Adding the viewer JavaScript file to your web page.
2. Defining the container `DIV`.
3. Creating and initializing the viewer.

All the steps above are the same as with fixed size embedding. Add the container `DIV` to the existing "holder" `DIV`. The following code is a complete example. You can see how the viewer size changes when the browser is resized, and how the viewer aspect ratio matches the asset.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/SpinViewer.js"></script>
<style type="text/css">
.holder {
  width: 40%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var spinViewer = new s7viewers.SpinViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/SpinSet_Sample",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  }
}).init();
</script>
</body>
</html>
```

The following examples page illustrates more real-life use cases of responsive design embedding with unrestricted height:

https://marketing.adobe.com/resources/help/en_US/s7/vlist/vlist.html

Flexible size embedding with width and height defined

In case of flexible-size embedding with width and height defined, the web page styling is different. That is, it provides both sizes to the "holder" DIV and centers it in the browser window. Also, the web page sets the size of the HTML and BODY element to 100%:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>
```

The remaining embedding steps are identical to responsive design embedding with unrestricted height. The resulting example is the following:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/SpinViewer.js"></script>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var spinViewer = new s7viewers.SpinViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Scene7SharedAssets/SpinSet_Sample",
    "serverurl":"http://s7dl.scene7.com/is/image/"
  }
}).init();
</script>
</body>
</html>
```

Embedding using Setter-based API

Instead of using JSON-based initialization it is possible to use setter-based API and no-args constructor. With that API constructor does not take any parameters and configuration parameters is specified using `setContainerId()`, `setParam()`, and `setAsset()` API methods with separate JavaScript calls.

The following example shows fixed size embedding with setter-based API:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/SpinViewer.js"></script>
<style type="text/css">
#s7viewer.s7spinviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var spinViewer = new s7viewers.SpinViewer();
spinViewer.setContainerId("s7viewer");
spinViewer.setParam("serverurl", "http://s7dl.scene7.com/is/image/");
spinViewer.setAsset("Scene7SharedAssets/SpinSet_Sample");
spinViewer.init();
</script>
</body>
</html>
```

Command reference – Configuration attributes

Configuration attributes documentation for Spin Viewer.

Any configuration command can be set in URL or using `setParam()`, or `setParams()`, or both, API methods. Any config attribute can be also specified in server-side configuration record.

Some configuration commands may be prefixed with the class name or instance name of corresponding Viewer SDK component. An instance name of the component is dynamic and depends on the ID of the viewer container DOM element passed to `setContainerId()` API method. Documentation includes an optional prefix for such commands. For example, `zoomstep` command is documented as follows:

```
[SpinView.|<containerId>_spinView].zoomstep
```

which means that you can use this command as:

- `zoomstep` (short syntax)
- `SpinView.zoomstep` (qualified with component class name)
- `cont_spinView.zoomstep` (qualified with component ID, assuming `cont` is the ID of the container element)

See also [Command reference common to all viewers – Configuration attributes](#)

closebutton

`closebutton=0|1`

0-1	Set to 1 to enable close button display, or set to 0 to hide close button.
-----	--

Properties

Optional.

Default

0

Example

closebutton=1

SpinView.doubleclick

[SpinView. | <containerId>_spinView.]doubleclick=none | zoom | reset | zoomReset

none zoom reset zoomReset	Configures the mapping of double-click/tap to spin actions. Setting to none disables double-click/tap spin. If set to zoom clicking the image spins in one spin step; CTRL+Click spins out one spin step. Setting to reset causes a single click on the image to reset the spin to the initial spin level. For zoomReset, reset is applied if the current spin factor is at or beyond the specified limit, otherwise spin is applied.
---------------------------------	---

Properties

Optional.

Default

reset on desktop computers; zoomReset on touch devices.

Example

doubleclick=zoom

SpinView.fmt

[SpinView. | <containerId>_spinView.]fmt=jpg | jpeg | png | png-alpha | gif | gif-alpha

jpg jpeg png png-alpha gif gif-alpha	Specifies the image format that the component uses for loading images from Image Server. If the specified format ends with -alpha, the component renders images as transparent content. For all other image formats the component treats images as opaque. The component has a white background by default. Therefore, to make it transparent, set the background-color CSS property to transparent.
--	--

Properties

Optional.

Default

jpeg

Example

fmt=png-alpha

SpinView.iconeffect

[SpinView.|<containerId>_spinView.]iconeffect=0|1[,count][,fade][,autoHide]

0 1	Enables the <code>iconeffect</code> to display on the top of the image when the image is in a reset state and it is suggestive of an available action to interact with the image.
<i>count</i>	Specifies the maximum number of times the <code>iconeffect</code> appears and reappears. A value of -1 indicates that the icon always reappears indefinitely.
<i>fade</i>	Specifies the duration of the show or hide animation, in seconds.
<i>autoHide</i>	Sets the number of seconds that the <code>iconeffect</code> stays fully visible before it auto hides. That is, the time after the fade-in animation is completed but before the fade out animation starts. A setting of 0 disables the auto-hide behavior.

Properties

Optional.

Default


1,1,0.3,3

Example

iconeffect=0

SpinView.iscommand

[SpinView.|<containerId>_spinView.]iscommand=*isCommand*

<i>iscommand</i>	<p>The Image Serving command string that is applied to spin image. If specified in the URL all occurrences of & and = must be HTTP-encoded as %26 and %3D, respectively.</p> <p> Note: Image sizing manipulation commands are not supported.</p>
------------------	--

Properties

Optional.

Default

None.

Example

When specified in the viewer URL:

```
iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000
```

When specified in the config data:

```
iscommand=op_sharpen=1&op_colorize=0xff0000
```

SpinView.lockdirection

```
[SpinView.|<containerId>_spinView.]lockdirection=0|1
```

0 1	<p>Specifies whether to allow a change in the spin direction in case of 2D spin set.</p> <p>When set to 1, the component identifies the primary drag or swipe direction (horizontal versus vertical) at the start of the gesture. After that, it maintains that direction until the gesture ends. For example, if the user starts a horizontal spin and then decides to continue their drag gesture in a vertical direction, the component does not perform a vertical spin; instead, it considers only the horizontal movement of the mouse or swipe.</p> <p>A value of 0 lets a user change the spin direction at any time during the gesture progress. The setting has no affect if the spin set is 1D.</p>
-----	--

Properties

Optional.

Default

0

Example

```
lockdirection=1
```

SpinView.maxloadradius

```
[SpinView.|<containerId>_spinView.]maxloadradius=value[,highRes]
```

<i>value</i>	<p>Represents the maximum number of frames to preload in each direction when the SpinView is idle. A value of -1 preloads all frames in the set. The preloaded frames are always seen at the original resolution that the SpinView was initially loaded.</p>
<i>highRes</i>	<p>Controls the quality of preloaded frames. When set to 1 frames load in high-quality, matching the size of the component. When set to 0 only the low-resolution preview tile is loaded.</p> <p>Preloading in high resolution improves the end-user experience, especially when auto-spin is enabled. At the same time, it results in slower start time and higher network consumption, so use with caution. When a high-resolution preload is used the preloaded frames are always at the original resolution that the component was initially loaded at.</p>

Properties

Optional.

Default

6,0

Example

```
maxloadradius=12,1
```

SpinView.autospin

```
[SpinView.|<containerId>_spinView.]maxloadradius=0|1[,duration][,direction][,spin_number]
```

0 1	Enables or disables the automatic spin animation. To achieve the best auto spin experience it is recommended that you preload all frames by setting <code>maxloadradius</code> to -1. Note, however, that this results in increased load time and higher bandwidth usage.
<i>duration</i>	The number of seconds per one full spin.
<i>direction</i>	The spin direction which is 0 for spinning east and 1 for spinning west.
<i>spin_number</i>	The number of full rotations done before autospin stops. The number is a floating point number. Set to -1 for an infinite auto spin.

Properties

Optional.

Default

0,1,1,1

Example

```
autospin=1,2,-1,1
```

SpinView.sensitivity

```
[SpinView.|<containerId>_spinView.]sensitivity=xSensitivity[, ySensitivity]
```

<i>xSensitivity</i> [, <i>ySensitivity</i>]	<p>Controls the sensitivity of the horizontal and vertical spin performed with a mouse drag or swipe.</p> <p><i>xSensitivity</i> sets how many full horizontal product rotations are made if the user drags their mouse horizontally from one side of the view to the other. For example, three means that the user sees three complete spins for one full drag gesture.</p>
---	--

	<p>Likewise, <code>ySensitivity</code> controls the sensitivity of the vertical spin. A value of 1 means that one full vertical drag or swipe changes the view angle from the top-most spin plane to the bottom-most (or vice versa).</p> <p>Setting a negative value for <code>ySensitivity</code> reverses the direction of the vertical spin.</p>
--	--

Properties

Optional.

Default

5, 2

Example

`sensitivity=4,-2`

SpinView.singleclick

`[SpinView.<containerId>_spinView.]singleclick=none|zoom|reset|zoomReset`

<code>none zoom reset zoomReset</code>	Configures the mapping of single-click/tap to zoom actions. Setting to <code>none</code> disables single-click/tap zoom. If set to <code>spin</code> clicking the image zooms in one zoom step; CTRL+Click zooms out one zoom step. Setting to <code>reset</code> causes a single click on the image to reset the zoom to the initial spin level. For <code>zoomReset</code> , reset is applied if the current zoom factor is at or beyond the specified limit, otherwise zoom is applied.
--	--

Properties

Optional.

Default

`zoomReset` on desktop computers; `none` on touch devices.

Example

`singleclick=zoom`

SpinView.transition

`[SpinView.<containerId>_spinView.]transition=time[,easing]`

<code>time</code>	Specifies the time in seconds that the animation for a single zoom step action takes.
<code>easing</code>	Creates an illusion of acceleration or deceleration which makes the transition appear more natural. You can set easing to one of the following: <ul style="list-style-type: none">0 (auto)

- 1 (linear)
- 2 (quadratic)
- 3 (cubic)
- 4 (quartic)
- 5 (quintic)

Auto mode always uses linear transition when elastic zoom is disabled (default). Otherwise, it fits one of the other easing functions based on the transition time. That is, the shorter the transition time the higher the easing function is used to accelerate the acceleration or deceleration effect.

Properties

Optional.

Default

0.5,0

Example

```
transition=2,2
```

SpinView.zoomstep

```
[SpinView.<containerId>_spinView.]zoomstep=step[,limit]
```

<i>step</i>	Configures the number zoom in and zoom out actions that are required to increase or decrease the resolution by a factor of two. The resolution change for each zoom action is 2 ¹ per step. Set to 0 to zoom to full resolution with a single zoom action.
<i>limit</i>	Specifies the maximum zoom resolution, relative to the full resolution image. The default is 1.0, which does not allow zooming beyond full resolution.

Properties

Optional.

Default

1,1

Example

```
zoomstep=2,3
```

ZoomView.enableHD

```
[ZoomView.<containerId>_zoomView.]enableHD=always|never|limit[,number]
```

<code>always never limit</code>	<p>Enable, limit or disable optimization for devices where <code>devicePixelRatio</code> is greater than 1, that is devices with high-density display like iPhone4 and similar devices. If active then the component limits the size of the IS image request as if the device only had a pixel ratio of 1 and that way reducing the bandwidth.</p> <p>See example below.</p>
<code>number</code>	<p>If using the limit setting, the component enables high pixel density only up to the specified limit.</p> <p>See example below.</p>

Properties

Optional.

Default

`limit,1500`

Example

The following are the expected results when you use this configuration attribute with the viewer, and the viewer size is 1000 x 1000:

If the set variable equals	Result
<code>always</code>	<p>The pixel density of the screen/device is always taken into account.</p> <ul style="list-style-type: none"> • If the screen pixel density = 1, then the requested image is 1000 x 1000. • If the screen pixel density = 1.5, then the requested image is 1500 x 1500. • If the screen pixel density = 2, then the requested image is 2000 x 2000.
<code>never</code>	<p>It always uses a pixel density of 1 and ignores the device's HD capability. Therefore, the requested image requested is always 1000 x 1000.</p>
<code>limit<number></code>	<p>A device pixel density is requested and served only if the resulting image is below the specified limit.</p> <p>The limit number applies to either the width or the height dimension.</p> <ul style="list-style-type: none"> • If the limit number is 1600, and the pixel density is 1.5, then the 1500 x 1500 image is served. • If the limit number is 1600, and the pixel density is 2, then the 1000 x 1000 image is served because the 2000 x 2000 image exceeds the limit. <p>Best practice: The limit number needs to work in conjunction with the company setting for maximum size image. Therefore, set the limit number to equal the company maximum image size setting.</p>

Javascript API reference for Spin Viewer

The main class of the Spin Viewer is `SpinViewer`. It is declared in the `s7viewers` namespace. This JavaScript API covers constructor, methods, and call backs of this particular class.

In all the following examples, `<instance>` stands for the actual name of the JavaScript viewer object that is instantiated from the `s7viewers.SpinViewer` class.

dispose

JavaScript API reference for Spin Viewer.

```
dispose()
```

Disposes this viewer instance by releasing all resources used by the viewer logic and deleting all inner objects and components created by the viewer in runtime.

The web page code should also delete the viewer instance variable as well to completely remove the viewer from the web browser memory.

If the web page code has registered event listeners directly on Viewer SDK components used by the viewer—or stored external references to such components—such listeners must be explicitly unregistered by the web page code, and such external component references must be deleted prior to calling `dispose()`.

Do not access the Viewer API any more after `dispose()` is called.

Parameters

None.

Returns

None.

Example

```
<instance>.dispose()
```

getComponent

JavaScript API reference for Spin Viewer

```
getComponent(componentId)
```

Returns a reference to the Viewer SDK component that is used by the viewer. The web page can use this method to extend or customize the behavior of the out-of-box viewer. Call this method only after the `initComplete` viewer callback has run, otherwise the component may not be created yet by the viewer logic.

Parameters

componentID – {String} an ID of the Viewer SDK component used by the viewer. This viewer supports the following component IDs:

Component ID	Viewer SDK component class name
parameterManager	s7sdk.ParameterManager
container	s7sdk.common.Container
mediaSet	s7sdk.set.MediaSet
spinView	s7sdk.set.SpinView
zoomInButton	s7sdk.common.ZoomInButton
zoomOutButton	s7sdk.common.ZoomOutButton
zoomResetButton	s7sdk.common.ZoomResetButton
fullScreenButton	s7sdk.common.FullScreenButton
closeButton	s7sdk.common.CloseButton
spinLeftButton	s7sdk.common.PanLeftButton
spinRightButton	s7sdk.common.PanRightButton

When working with SDK APIs it is important to use a correct, fully qualified SDK namespace as described in [Viewer SDK](#).

See the Viewer SDK API documentation for more information about a particular component.

Returns

{Object} a reference to Viewer SDK component. The method returns null if the componentId is not a supported viewer component or if the component was not yet created by the viewer logic.

Example

```
<instance>.setHandlers({  
  "initComplete":function() {  
    var spinView = <instance>.getComponent("spinView");  
  }  
})
```

init

JavaScript API reference for Spin Viewer.

init()

Starts the initialization of the Spin Viewer. By this time the container DOM element must be created so that the viewer code can find it by its ID.

If the container element is not a part of the web page layout just yet (for example, it may be hidden using `display:none` style assigned to it), the viewer suspends its initialization process until the moment when the web page brings the container element back to the layout. When this occurs, the viewer load automatically resumes.

Call this method only once during viewer life cycle; subsequent calls are ignored.

Parameters

None.

Returns

{Object} A reference to the viewer instance.

Example

```
<instance>.init()
```

setAsset

JavaScript API reference for Spin Viewer.

setAsset(asset)

asset	<p>{String} new asset id, single or multi-dimensional spin set with optional Image Serving modifiers appended after ?.</p> <p>Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.</p>
-------	---

Sets the new asset. You can call this parameter at any time, either before or after `init()`. If it is called after `init()`, the viewer swaps the asset at runtime.

See also [init](#).

Returns

None.

Example

Single reference to a spin set that is defined in a catalog:

```
<instance>.setAsset("Scene7SharedAssets/SpinSet_Sample")
```

Explicit spin set:

```
<instance>.setAsset('Scene7Assets/Par1,Scene7Assets/Par2,Scene7Assets/Par3,Scene7Assets/Par4,Scene7Assets/Par5,Scene7Assets/Par6,Scene7Assets/Par7,Scene7Assets/Par8')
```

Explicit multi-dimensional spin set:

```
<instance>.setAsset('Scene7Assets/Par1,Scene7Assets/Par2,Scene7Assets/Par3,Scene7Assets/Par4,Scene7Assets/Par5,Scene7Assets/Par6,Scene7Assets/Par7,Scene7Assets/Par8')
```

setContainerId

JavaScript API reference for Spin Viewer.

setContainerId(containerId)

Sets the ID of the DOM container (normally a DIV) into which the viewer is inserted. It is not necessary to have the container element created by the time this method is called. However, the container must exist when `init()` is run. It must be called before `init()`.

This method is optional if the viewer configuration information is passed with `config` JSON object to the constructor.

<i>containerId</i>	{string} ID of container.
--------------------	---------------------------

Returns

None.

Example

```
<instance>.setContainerId("s7viewer");
```

setHandlers

JavaScript API reference for Spin Viewer

`setHandlers(handlers)`

Specifies zero or more callback handlers. A call to this method fully overwrites event handlers that were previously assigned for that viewer instance. Must be called before `init()`.

Parameter

<i>handlers</i>	<p>{Object} JSON object with viewer event callbacks, where the property name is the name of the supported viewer event and the property value is a JavaScript function reference to an appropriate callback.</p> <p>See Event callbacks for more information about viewer events.</p>
-----------------	---

Returns

None.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    console.log("init complete");
  }
})
```

setLocalizedTexts

JavaScript API reference for Spin Viewer.

`setLocalizedTexts(localizationInfo)`

<i>localizationInfo</i>	<p>{Object} JSON object with localization data.</p> <p>See Localization of user interface elements for more information.</p>
-------------------------	--

	See also the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.
--	--

Sets localization SYMBOL values for one or more locales. This parameter must be called before `init()`.

See also [init](#).

Returns

None.

Example

```
<instance>.setLocalizedTexts({"en":{"CloseButton.TOOLTIP":"Close"},"fr":{"CloseButton.TOOLTIP":"Fermer"},"defaultLocale":"en"})
```

setParam

JavaScript API reference for Spin Viewer.

```
setParam(name, value)
```

<i>name</i>	{string} name of parameter.
<i>value</i>	{string} value of parameter. The value cannot be percent-encoded.

Sets the viewer parameter to a specified value. The parameter is either a viewer-specific configuration option or a software development kit modifier. This parameter is called before `init()`.

This method is optional if the viewer configuration information is passed with `config` JSON object to the constructor.

See also [init](#).

Returns

None.

Example

```
<instance>.setParam("style", "customStyle.css")
```

setParams

JavaScript API reference for Spin Viewer.

```
setParams(params)
```

<i>params</i>	{string} name=value parameter pairs separated with &.
---------------	---

Sets one or more parameters to a given value. The method argument syntax is identical to a URL query string. That is, it represents name=value pairs separated with &. Just as in a query string, the names and values are percent-encoded using UTF8. Before you call `init()`, this parameter must be called.

This method is optional if the viewer configuration information is passed with `config` JSON object to the constructor.

See also [init](#).

Returns

None.

Example

```
<instance>.setParams( "SpinView.zoomstep=2,3&SpinView.iscommand=op_sharpen%3d1" )
```

SpinViewer

JavaScript API reference for Spin Viewer.

```
SpinViewer([config])
```

Constructor, creates a new Spin Viewer instance.

Parameters

<i>config</i>	<p>{Object} optional JSON configuration object, allows all the viewer settings to pass to the constructor and avoid calling individual setter methods. Contains the following properties:</p> <ul style="list-style-type: none">• containerId – {String} ID of the DOM container (normally a <code>DIV</code>) that the viewer is inserted into. It is not necessary to have the container element created by the time this method is called. However, the container must exist when <code>init()</code> is run. Required.• params – {Object} JSON object with viewer configuration parameters where the property name is either a viewer-specific configuration option or an SDK modifier, and the value of that property is a corresponding settings value. Required.• handlers – {Object} JSON object with viewer event callbacks, where the property name is the name of supported viewer event, and the property value is a JavaScript function reference to an appropriate callback. Optional. See Event callbacks for more information about viewer events.• localizedTexts – {Object} JSON object with localization data. Optional. See Localization of user interface elements for more information. <p>See also the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
---------------	--

Returns

None.

Example

```
var spinViewer = new s7viewers.SpinViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/SpinSet_Sample",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  },
  "handlers": {
```

```

    "initComplete":function() {
        console.log("init complete");
    },
    "localizedTexts":{
        "en":{
            "CloseButton.TOOLTIP":"Close"
        },
        "fr":{
            "CloseButton.TOOLTIP":"Fermer"
        },
        defaultLocale:"en"
    }
  });

```

Event callbacks

The viewer supports JavaScript event callbacks that the web page uses to track the viewer initialization process or runtime behavior.

Callback handlers are assigned by passing event names and corresponding handler functions with the `handlers` property to `config` JSON object in the viewer's constructor. Alternatively, it is possible to use `setHandlers()` API method.

Supported viewer events include the following:

- `initComplete` – triggers when viewer initialization is complete and all internal components are created, so that it is possible to use `getComponent()` API. The callback handler does not take any arguments.
- `trackEvent` – triggers each time an event occurs inside the viewer which may be handled by an event tracking system, such as Adobe Analytics. The callback handler takes the following arguments:
 - `objID` {String} not currently used.
 - `compClass` {String} not currently used.
 - `instName` {String} an instance name of the Viewer SDK component that triggered the event.
 - `timeStamp` {Number} event time stamp.
 - `eventInfo` {String} event payload.

See also [SpinViewer](#) and [setHandlers](#).

Customizing Spin Viewer

All visual customization and most behavior customization for the Spin Viewer is done by creating a custom CSS.

The suggested workflow is to take the default CSS file for the appropriate viewer, copy it to a different location, customize it, and specify the location of the customized file in the `style=` command.

Default CSS files can be found at the following location:

```
<s7_viewers_root>/html5/SpinViewer_light.css
```

The viewer is supplied with out-of-the-box CSS files, for "light" and "dark" color schemes. The "light" version is used by default, but you can switch to the "dark" version by using the following standard CSS:

```
<s7_viewers_root>/html5/SpinViewer_dark.css
```

Custom CSS file must contain the same class declarations as the default one. If a class declaration is omitted, the viewer does not function properly because it does not provide built-in default styles for user interface elements.

Alternative way to provide custom CSS rules is to use embedded styles directly on the web page or in one of linked external CSS rules.

When creating custom CSS keep in mind that the viewer assigns `.s7spinviewer` class to its container DOM element. If you are using external CSS file passed with `style=` command, use `.s7spinviewer` class as parent class in descendant selector for your CSS rules. If you are doing embedded styles on the web page, additionally qualify this selector with an ID of the container DOM element as follows:

```
#<containerId>.s7spinviewer
```

Building responsive designed CSS

It is possible to target different devices and embedding sizes in CSS to make your content display differently, depending on a user's device or a particular web page layout. This includes, but is not limited to, different web page layouts, user interface element sizes, and artwork resolution.

The viewer supports two methods for creating responsive designed CSS: CSS markers and standard CSS media queries. You can use these methods independently or together.

CSS markers

To assist in creating responsive designed CSS, the viewer supports CSS markers which special CSS classes dynamically assigned to the top-level viewer container element based on the run-time viewer size and the input type used on the current device.

The first group of CSS markers includes `.s7size_large`, `.s7size_medium`, and `.s7size_small` classes. They are applied based on the runtime area of the viewer container. That is, if the viewer area is equal to or bigger than the size of a common desktop monitor `.s7size_large` is used; if the area is close in size to a common tablet device `.s7size_medium` is assigned. For areas similar to mobile phone screens `.s7size_small` is set. The primary purpose of these CSS markers is to create different user interface layouts for different screens and viewer sizes.

The second group of CSS markers includes `.s7mouseinput` and `.s7touchinput`. `.s7touchinput` is set if the current device has touch input capabilities; otherwise, `.s7mouseinput` is used. These markers are intended to create user interface input elements with different screen sizes for different input types, because normally touch input requires larger elements. In case the device has both mouse input and touch capabilities, `.s7touchinput` is set and the viewer renders a touch-friendly user interface.

The following sample CSS sets the zoom in button size to 28 x 28 pixels on systems with mouse input, and 56 x 56 pixels on touch devices. In addition, it hides the button completely if the viewer size becomes really small:

```
.s7spinviewer.s7mouseinput .s7zoominbutton {
    width:28px;
    height:28px;
}
.s7spinviewer.s7touchinput .s7zoominbutton {
    width:56px;
    height:56px;
}
.s7spinviewer.s7size_small .s7zoominbutton {
    visibility:hidden;
}
```

To target devices with a different pixel density, use CSS media queries. The following media query block would contain CSS that is specific to high density screens:

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5)
{
}
```

Using CSS markers is the most flexible way of building responsive designed CSS as it allows you to target not only device screen size but actual viewer size, which may be useful for responsive design page layouts.

Use the default viewer CSS file as an example of a CSS markers approach.

CSS media queries

Device sensing can also be done using pure CSS media queries. Everything enclosed within a given media query block is applied only when it is run on a corresponding device.

When applied to Mobile Viewers, use four CSS media queries, defined in your CSS in the following order:

1. Contains only rules specific for all touch devices.

```
@media only screen and (max-device-width:13.5in) and (max-device-
height:13.5in) and (max-device-width:799px),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in)
and (max-device-height:799px)
{
}
```

2. Contains only rules specific for tablets with high resolution screens.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px) and (-webkit-min-device-pixel-ratio:1.5),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px) and (-webkit-min-device-pixel-ratio:1.5)
{
}
```

3. Contains only rules specific for all mobile phones.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in)
{
}
```

4. Contains only rules specific for mobile phones with high resolution screens.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in) and
(-webkit-min-device-pixel-ratio: 1.5),
only screen and (device-width:720px) and (device-height:1280px) and
(-webkit-device-pixel-ratio: 2),
only screen and (device-width:1280px) and (device-height:720px) and
(-webkit-device-pixel-ratio: 2)
{
}
```

Using a media queries approach, you should organize CSS with device sensing as follows:

- First, the desktop-specific section defines all properties that are either desktop-specific or common to all screens.
- And second, the four media queries go in the order defined above and provide CSS rules that are specific for the corresponding device type.

There is no need to duplicate the entire viewer CSS in each media query. Only properties that are specific to given devices are redefined inside a media query.

CSS Sprites

Many viewer user interface elements are styled using bitmap artwork and have more than one distinct visual state. A good example is a button that normally has at least 3 different states: "up", "over", and "down". Each state requires its own bitmap artwork assigned.

With a classic approach to styling, the CSS would have a separate reference to individual image file on the server for each state of the user interface element. The following is a sample CSS for styling zoom-in button:

```
.s7spinviewer.s7mouseinput .s7zoominbutton[state='up'] {
background-image:url(images/v2/ZoomInButton_dark_up.png);
}
.s7spinviewer.s7mouseinput .s7zoominbutton[state='over'] {
background-image:url(images/v2/ZoomInButton_dark_over.png);
}
.s7spinviewer.s7mouseinput .s7zoominbutton[state='down'] {
background-image:url(images/v2/ZoomInButton_dark_down.png);
}
.s7spinviewer.s7mouseinput .s7zoominbutton[state='disabled'] {
background-image:url(images/v2/ZoomInButton_dark_disabled.png);
}
```

The drawback to this approach is that the end user experiences flickering or delayed user interface response when the element is interacted with for the first time. This action occurs because the image artwork for the new element state is not yet downloaded. Also, this approach may have a slight negative impact on performance because of an increase in the number of HTTP calls to the server.

CSS sprites is a different approach where image artwork for all element states is combined into a single PNG file called a "sprite". Such "sprite" has all visual states for the given element positioned one after another. When styling a user interface element with sprites the same sprite image is referenced for all different states in the CSS. Also, the `background-position` property is used for each state to specify which part of the "sprite" image is used. You can structure a "sprite" image in any suitable way. Viewers normally have it vertically stacked. Below is a "sprite"-based example of styling the same zoom-in button from above:

```
.s7spinviewer .s7zoominbutton[state] {
background-image: url(images/v2/ZoomInButton_dark_sprite.png);
}
.s7spinviewer.s7mouseinput .s7zoominbutton[state='up'] {
background-position: -84px -560px;
}
.s7spinviewer.s7mouseinput .s7zoominbutton[state='over'] {
background-position: -56px -560px;
}
.s7spinviewer.s7mouseinput .s7zoominbutton[state='down'] {
background-position: -28px -560px;
}
.s7spinviewer.s7mouseinput .s7zoominbutton[state='disabled'] {
background-position: -0px -560px;
}
```

General styling notes and advice

- When customizing the viewer user interface with CSS the use of the `!IMPORTANT` rule is not supported to style viewer elements. In particular, `!IMPORTANT` rule should not be used to override any default or run-time styling provided by the viewer or Viewer SDK. The reason is that it may affect the behavior of proper components. Instead, you should use CSS selectors with the proper specificity to set CSS properties that are documented in this reference guide.
- All paths to external assets within CSS are resolved against the CSS location, not the viewer HTML page location. Be aware of this rule when you copy the default CSS to a different location. Either copy the default assets as well or update paths within the custom CSS.
- The preferred format for bitmap artwork is PNG.
- Bitmap artwork is assigned to user interface elements using the `background-image` property.
- The `width` and `height` properties of a user interface element define its logical size. The size of the bitmap passed to `background-image` does not affect logical size.

- To use the high pixel density of high-resolution screens like Retina, specify bitmap artwork twice as large as the logical user interface element size. Then, apply the `-webkit-background-size: contain` property to scale the background down to the logical user interface element size.
- To remove a button from the user interface, add `display: none` to its CSS class.
- You can use various formats for color value that CSS supports. If you need transparency, use the format `rgba(R,G,B,A)`. Otherwise, you can use the format `#RRGGBB`.

Common User Interface Elements

The following is user interface elements reference documentation that applies to Mixed Media Viewer:

Close button

Clicking or tapping this button closes the containing web page. This button only appears if the `closebutton` parameter is set to 1. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7spinviewer .s7closebutton
```

CSS property	Description
<code>top</code>	Position from the top border, including padding.
<code>right</code>	Position from the right border, including padding.
<code>left</code>	Position from the left border, including padding.
<code>bottom</code>	Position from the bottom border, including padding.
<code>width</code>	Width of the button.
<code>height</code>	Height of the button.
<code>background-image</code>	The image that is displayed for a given button state.
<code>background-position</code>	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a close button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7spinviewer .s7closebutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7spinviewer .s7closebutton [state='up'] {
background-image:url(images/v2/CloseButton_dark_up.png);
}
.s7spinviewer .s7closebutton [state='over'] {
background-image:url(images/v2/CloseButton_dark_over.png);
}
.s7spinviewer .s7closebutton [state='down'] {
background-image:url(images/v2/CloseButton_dark_down.png);
}
.s7spinviewer .s7closebutton [state='disabled'] {
background-image:url(images/v2/CloseButton_dark_disabled.png);
}
```

Focus highlight

Input focus highlight displayed around focused viewer UI element is controlled with the CSS class selector.

CSS properties

The appearance is controlled with the following CSS class selector:

```
.s7spinviewer *:focus
```

CSS property	Description
outline	Focus highlight style.

Example – to disable the default browser focus highlight for all viewer user interface elements, add the following CSS selector to viewer's style sheet:

```
.s7spinviewer *:focus {
outline: none;
}
```

Full screen button

This button causes the viewer to enter or exit full screen mode when clicked by the user. This button is not display if the viewer works in pop-up mode and the system does not support native full screen. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7spinviewer .s7fullscreenbutton
```

CSS property	Description
top	Position from the top border, including padding.

CSS property	Description
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the "full screen" state and *selected='false'* corresponds to the "normal" state.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a full screen button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states when selected or not selected:

```
.s7spinviewer .s7fullscreenbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7spinviewer .s7fullscreenbutton [selected='false'][state='up'] {
background-image:url(images/enterFullBtn_up.png);
}
.s7spinviewer .s7fullscreenbutton [selected='false'][state='over'] {
background-image:url(images/enterFullBtn_over.png);
}
.s7spinviewer .s7fullscreenbutton [selected='false'][state='down'] {
background-image:url(images/enterFullBtn_down.png);
}
.s7spinviewer .s7fullscreenbutton [selected='false'][state='disabled'] {
background-image:url(images/enterFullBtn_disabled.png);
}
.s7spinviewer .s7fullscreenbutton [selected='true'][state='up'] {
background-image:url(images/exitFullBtn_up.png);
}
.s7spinviewer .s7fullscreenbutton [selected='true'][state='over'] {
background-image:url(images/exitFullBtn_over.png);
}
.s7spinviewer .s7fullscreenbutton [selected='true'][state='down'] {
background-image:url(images/exitFullBtn_down.png);
}
.s7spinviewer .s7fullscreenbutton [selected='true'][state='disabled'] {
```



```
background-image:url(images/exitFullBtn_disabled.png); }
}
```

Icon effect

The spin indicator is overlaid on the main view area. It is displayed when the image is in a reset state and it also depends on the `iconeffect` parameter.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7spinviewer .s7spinview .s7iconeffect
```

CSS property	Description
background-image	Spin indicator artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
width	Spin indicator width.
height	Spin indicator height.

Spin indicator supports the `state` attribute selector which is set to `spin_1D` in case of single-dimensional spin set and to `spin_2D` in case of multi-dimensional spin set.

Example – to set up a 100 x 100 pixels zoom indicator.

```
.s7spinviewer .s7spinview .s7iconeffect {
  width: 100px;
  height: 100px;
}
.s7spinviewer .s7spinview .s7iconeffect[state="spin_1D"] {
  background-image: url(images/spinIcon_1D.png);
}
.s7spinviewer .s7spinview .s7iconeffect[state="spin_2D"] {
  background-image: url(images/spinIcon_2D.png);
}
```

Main viewer area

The main view area is the area occupied by the spin image. It usually sets to fit the available device screen when no size is specified.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7spinviewer
```

CSS property	Description
width	The width of the viewer.

CSS property	Description
height	The height of the viewer.
background-color	Background color in hexadecimal format.

Example – to set up a viewer with a white background (#FFFFFF) and make its size 512 x 288 pixels.

```
.s7spinviewer {
  background-color: #FFFFFF;
  width: 512px;
  height: 288px;
}
```

Spin left button

Clicking or tapping this button spins the image to the left in main view. This button is not displayed on mobile phones in order to save screen real estate. Also, the button is hidden when a multi-dimensional spin set is used. You can size, skin, and position the button using CSS.

CSS properties of the spin buttons

The button is added to an internal container that is DIV controlled with the CSS class selector:

```
.s7spinviewer .s7spinbuttons
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.

The appearance of this button inside the container is controlled with the CSS class selector:

```
.s7spinviewer .s7spinbuttons .s7panleftbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.

CSS property	Description
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a spin left button that is 28 x 28 pixels, that is positioned on the left edge of inner container, and that displays a different image for each of the four different button states:

```
.s7spinviewer .s7spinbuttons .s7panleftbutton {
  position:absolute;
  left: 0px;
  width:28px;
  height:28px;
  background-size:contain;
}
.s7spinviewer .s7spinbuttons .s7panleftbutton[state='up'] {
  background-image:url(images/v2/SpinLeftButton_light_up.png);
}
.s7spinviewer .s7spinbuttons .s7panleftbutton[state='over'] {
  background-image:url(images/v2/SpinLeftButton_light_over.png);
}
.s7spinviewer .s7spinbuttons .s7panleftbutton[state='down'] {
  background-image:url(images/v2/SpinLeftButton_light_down.png);
}
.s7spinviewer .s7spinbuttons .s7panleftbutton[state='disabled'] {
  background-image:url(images/v2/SpinLeftButton_light_disabled.png);
}
```

Spin right button

Clicking or tapping this button spins the image to the right in main view. This button is not displayed on mobile phones in order to save screen real estate. Also, the button is hidden when a multi-dimensional spin set is used. You can size, skin, and position the button using CSS.

CSS properties of the spin buttons

The button is added to an internal container that is DIV controlled with the CSS class selector:

```
.s7spinviewer .s7spinbuttons
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.

The appearance of this button inside the container is controlled with the CSS class selector:

```
.s7spinviewer .s7spinbuttons .s7panrightbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a spin right button that is 28 x 28 pixels, that is positioned on the right edge of inner container, and that displays a different image for each of the four different button states:

```
.s7spinviewer .s7spinbuttons .s7panrightbutton {
  position:absolute;
```

```

right: 0px;
width:28px;
height:28px;
background-size:contain;
}
.s7spinviewer .s7spinbuttons .s7panrightbutton[state='up'] {
background-image:url(images/v2/SpinRightButton_light_up.png);
}
.s7spinviewer .s7spinbuttons .s7panrightbutton[state='over'] {
background-image:url(images/v2/SpinRightButton_light_over.png);
}
.s7spinviewer .s7spinbuttons .s7panrightbutton[state='down'] {
background-image:url(images/v2/SpinRightButton_light_down.png);
}
.s7spinviewer .s7spinbuttons .s7panrightbutton[state='disabled'] {
background-image:url(images/v2/SpinRightButton_light_disabled.png);
}

```

Spin view

Main view consists of the spin image.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7spinviewer .s7spinview
```

CSS property	Description
background-color	Background color in hexadecimal format of the main view.

Example – to make the main view transparent.

```

.s7spinviewer .s7spinview {
background-color: transparent;
}

```

Tooltips

On desktop systems some user interface elements like buttons have tooltips that are displayed on mouse hover.

CSS properties of the main viewer area

The appearance of tooltips is controlled with the following CSS class selector:

```
.s7tooltip
```

CSS property	Description
border-radius	Background border radius.
border-color	Background border color.
background-color	Background color.
color	Text color.

CSS property	Description
font-family	Text font name.
font-size	Text font size.



Note: In case tooltip styles are customized from within the embedding web page, all properties have to contain !IMPORTANT rule. This is not necessary if tooltips are customized in the viewer's CSS file.

Example – to set up tooltips that have a grey border with 3px corner radius, black background and white text written with Arial, 11 pixels size:

```
.s7tooltip {
border-radius: 3px 3px 3px 3px;
border-color: #999999;
background-color: #000000;
color: #FFFFFF;
font-family: Arial, Helvetica, sans-serif;
font-size: 11px;
}
```

Zoom in button

Clicking or tapping this button zooms in on an image in the main view. This button does not display on mobile phones in order to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7spinviewer .s7zoominbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used.

CSS property	Description
	See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom in button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7spinviewer .s7zoominbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7spinviewer .s7zoominbutton [state='up'] {
background-image:url(images/v2/ZoomInButton_dark_up.png);
}
.s7spinviewer .s7zoominbutton [state='over'] {
background-image:url(images/v2/ZoomInButton_dark_over.png);
}
.s7spinviewer .s7zoominbutton [state='down'] {
background-image:url(images/v2/ZoomInButton_dark_down.png);
}
.s7spinviewer .s7zoominbutton [state='disabled'] {
background-image:url(images/v2/ZoomInButton_dark_disabled.png);
}
```

Zoom out button

Clicking or tapping this button zooms out on an image in the main view. This button does not display on mobile phones in order to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7spinviewer .s7zoomoutbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.

CSS property	Description
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom out button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7spinviewer .s7zoomoutbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7spinviewer .s7zoomoutbutton [state='up'] {
background-image:url(images/v2/ZoomOutButton_dark_up.png);
}
.s7spinviewer .s7zoomoutbutton [state='over'] {
background-image:url(images/v2/ZoomOutButton_dark_over.png);
}
.s7spinviewer .s7zoomoutbutton [state='down'] {
background-image:url(images/v2/ZoomOutButton_dark_down.png);
}
.s7spinviewer .s7zoomoutbutton [state='disabled'] {
background-image:url(images/v2/ZoomOutButton_dark_disabled.png);
}
```

Zoom reset button

Clicking or tapping this button resets an image in the main view. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7spinviewer .s7zoomresetbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.

CSS property	Description
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a zoom reset button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7spinviewer .s7zoomresetbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7spinviewer .s7zoomresetbutton [state='up'] {
background-image:url(images/v2/ZoomResetButton_dark_up.png);
}
.s7spinviewer .s7zoomresetbutton [state='over'] {
background-image:url(images/v2/ZoomResetButton_dark_over.png);
}
.s7spinviewer .s7zoomresetbutton [state='down'] {
background-image:url(images/v2/ZoomResetButton_dark_down.png);
}
.s7spinviewer .s7zoomresetbutton [state='disabled'] {
background-image:url(images/v2/ZoomResetButton_dark_disabled.png);
}
```

Support for Adobe Analytics tracking

The Spin Viewer supports Adobe Analytics tracking out of the box.

Out-of-the-box tracking

The Spin Viewer supports Adobe Analytics tracking out-of-the-box.

To enable tracking, pass the proper company preset name as `config2` parameter.

The viewer also sends a single tracking HTTP request to the configured Image Server with the viewer type and version information.

Custom tracking

To integrate with third-party analytics systems, it is necessary to listen to the `trackEvent` viewer callback and process the `eventInfo` argument of the callback function, as necessary. The following code is an example of such handler function:

```
var spinViewer = new s7viewers.SpinViewer({
  "containerId":"s7viewer",
```

```

"params":{
  "asset":"Scene7SharedAssets/SpinSet_Sample",
  "serverurl":"http://s7dl.scene7.com/is/image/"
},
"handlers":{
  "trackEvent":function(objID, compClass, instName, timeStamp, eventInfo) {
    //identify event type
    var eventType = eventInfo.split(",")[0];
    switch (eventType) {
      case "LOAD":
        //custom event processing code
        break;
      //additional cases for other events
    }
  }
}
});

```

The viewer tracks the following SDK user events:

SDK user event	Sent when...
LOAD	viewer is loaded first.
SWAP	an asset is swapped in the viewer using the <code>setAsset()</code> API.
ZOOM	an image is zoomed.
PAN	an image is panned.
SPIN	a spin is performed.

Localization of user interface elements

Certain content that the Spin Viewer displays is subject to localization, including zoom buttons and a full screen button.

Every textual content in the viewer that can be localized is represented by a special Viewer SDK identifier called SYMBOL. Any SYMBOL has a default associated text value for the English locale ("en") supplied with the out-of-the-box viewer. It can also have user-defined values set for as many locales as needed.

When the viewer starts, it checks the current locale to see if there is a user-defined value for each supported SYMBOL for the locale. If there is, it uses the user-defined value; otherwise, it falls back to the out-of-the-box default text.

User-defined localization data can be passed to the viewer as a localization JSON object. Such object contains the list of supported locales, the SYMBOL text values for each locale, and the default locale.

An example of such a localization object is the following:

```

{
  "en":{
    "CloseButton.TOOLTIP":"Close",
    "ZoomInButton.TOOLTIP":"Zoom In"
  },
  "fr":{
    "CloseButton.TOOLTIP":"Fermer",
    "ZoomInButton.TOOLTIP":"Agrandir"
  },
}

```

```
defaultLocale: "en"
}
```

In the above example, the localization object defines two locales ("en" and "fr") and provides localization for two user interface elements in each locale.

The web page code should pass the localization object to the viewer constructor, as a value of `localizedTexts` field of the configuration object. An alternative option is to pass the localization object by calling the `setLocalizedTexts(localizationInfo)` method.

The following SYMBOLs are supported:

SYMBOL	Tooltip for the...
<code>CloseButton.TOOLTIP</code>	Close button.
<code>ZoomInButton.TOOLTIP</code>	Zoom in button.
<code>ZoomOutButton.TOOLTIP</code>	Zoom out button.
<code>ZoomResetButton.TOOLTIP</code>	Zoom reset button.
<code>FullScreenButton.TOOLTIP_SELECTED</code>	Full screen button in normal state.
<code>FullScreenButton.TOOLTIP_UNSELECTED</code>	Full screen button in full screen state.
<code>PanLeftButton.TOOLTIP</code>	Spin left button.
<code>PanRightButton.TOOLTIP</code>	Spin right button.

Full screen support

The viewer supports full screen operation mode.

On modern desktop browsers, except Internet Explorer 10 and older, and on some touch devices, the viewer uses "native" full screen mode. This mode means that the entire device screen is occupied by the viewer content.

On iOS devices and on older Internet Explorer browsers, the viewer uses "simulated" full screen mode instead. In this mode, the viewer simply resizes to take the full area of the web browser window. Also, the web browser Chrome and other windows are still visible on the screen.

An end user enters and leaves full screen mode by pressing the Full Screen button in the viewer user interface. When "native" full screen mode is used on desktop, it is also possible to exit it by pressing **Esc**.

Viewer SDK namespace

The viewer is built of many Viewer SDK components. In most cases, the web page does not need to interact with SDK components API directly; all common needs are covered in the viewer API itself.

However, some advanced use cases require that the web page obtain a reference to an inner SDK component using the `getComponent()` viewer API and then use all the flexibility of the APIs of SDK itself.

The namespace that is used to load and initialize SDK components by the viewer depends on the environment in which the viewer is operating. If the viewer is running in AEM (Adobe Experience Manager), the viewer loads SDK components into `s7viewers.s7sdk` namespace. Likewise, the viewer served from Scene7 Publishing System loads the SDK into `s7classic.s7sdk`.

In either case, the namespace used by the SDK inside the viewer has either `s7viewers` or `s7classic` as the prefix. And, it is different from the plain `s7sdk` namespace used in the SDK User Guide or SDK API documentation. For that reason, it is important to use a fully qualified SDK namespace when you write custom application code that communicates with internal viewer components.

For example, if you plan to listen to `StatusEvent.NOTF_VIEW_READY` event and the viewer is served from AEM, the fully qualified event type is `s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY`, and the event listener code looks similar to the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var spinView = <instance>.getComponent("spinView");
    spinView.addEventListener(s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});
```

The same code for the viewer served from Scene7 Publishing System looks like the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var spinView = <instance>.getComponent("spinView");
    spinView.addEventListener(s7classic.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});
```

Video

The Video Viewer is a video player that plays streaming and progressive video encoded in the H.264 format. It is delivered from Scene7 Publishing System or AEM Dynamic Media.

See [System requirements](#).

Both single video and Adaptive Video Sets are supported. Additionally, the viewer supports working with progressive video and HLS streams hosted on external locations. It is designed to work on both desktop and mobile Web browsers that support HTML5 video. This viewer also supports optional closed captions that are displayed on top of video content, video chapter navigation, and social media sharing tools.

The Video Viewer uses HTML5 streaming video playback in HLS format in its default configuration whenever the underlying system supports it. On systems that do not support HTML5 streaming the viewer falls back to HTML5 progressive video delivery. Finally, the viewer still supports flash-based streaming video playback if requested, using the `VideoPlayer.playback` configuration parameter,

Viewer type 506.

Demo URL

https://s7d9.scene7.com/s7viewers/html5/VideoViewer.html?asset=Scene7SharedAssets/Glacier_Climber_MP4

Using Video Viewer

Video Viewer represent a main JavaScript file and a set of helper files—a single JavaScript include with all Viewer SDK components used by this particular viewer, assets, and CSS—downloaded by the viewer in runtime.

You can use the Video Viewer in pop-up mode using production-ready HTML page provided with IS-Viewers. Or, you can use the viewer in embedded mode, where it is integrated into a target web page using the documented API.

The task of configuring and skinning the viewer is similar to other viewers. All skinning is achieved by way of custom CSS.

See [Command reference common to all viewers – Configuration attributes](#) and [Command reference common to all Viewers – URL](#)

Interacting with Video Viewer

Video Viewer provides a set of standard user interface controls for video playback, like a play/pause button, video scrubber video time bubble, played time/total time indicator, volume control, full screen button, and closed caption toggle. All these controls are grouped into a control bar at the bottom of the viewer user interface.

On touch devices, volume control is hidden from the user interface, because it is only possible to control volume using the hardware buttons.

When the viewer operates in pop-up mode, the full screen button is not available in the user interface.

It is possible to navigate the content of a video quickly when video chaptering is activated. Video chapters are displayed as markers in the video scrubber track and show the chapter title and associated description on a mouse roll over or with a single tap on touch systems. Users can seek to a particular chapter by clicking on a chapter marker or tapping on the chapter description bubble.

The viewer supports both touch input and mouse input on Windows devices with touch screen and mouse. This support, however, is limited to Chrome, Internet Explorer 11, and Edge web browsers only.

This viewer is fully keyboard accessible.

See [Keyboard accessibility and navigation](#).

Social media sharing tools with Video Viewer

The Video Viewer supports social media sharing tools They are available as a single button in the user interface which expands into a sharing toolbar when the user clicks or taps on it.

The sharing toolbar contains an icon for each type of sharing channel supported such as Facebook, Twitter, email share, embed code share and link share. When email share, embed share, or link share tools are activated, the viewer displays a modal dialog box with a corresponding data entry form. When Facebook or Twitter are called, the viewer redirects the user to a standard sharing dialog box from a social media service. Also when a sharing tool is activated video playback is paused automatically.

Sharing tools are not available in full screen mode because of web browser security restrictions.

Embedding Video Viewer

Different web pages have different needs for viewer behavior. Sometimes a web page provides a link that, when clicked opens the viewer in a separate browser window. In other cases, it is necessary to embed the viewer directly on the hosting page. In the latter case the web page may have a static page layout, or use responsive design that displays differently on different devices or for different browser window sizes. To accommodate these needs, the viewer supports three primary operation modes: popup, fixed size embedding, and responsive design embedding.

Embedding multiple videos on the same page is supported on tablet and mobile devices. In a majority of cases, only one video can be played at a time. When a user starts playing one video, and then tries to play another video, the first video is automatically

paused. The video that was auto-paused remembers its current playback time, so the user can always get back to it and resume play. The only exception this rule is in Chrome browser on Android 4.x devices, which can play videos in parallel.

About pop-up mode

In pop-up mode the viewer is opened in a separate web browser window or tab. It takes the entire browser window area and adjusts in case the browser is resized or device orientation is changed.

This mode is the most common for mobile devices. The web page loads the viewer using `window.open()` JavaScript call, properly configured A HTML element, or any other suitable method.

It is recommended that you use an out-of-box HTML page for pop-up operation mode. It is called `VideoViewer.html` and it is located under the `html5/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/VideoViewer.html
```

You can achieve visual customization by applying custom CSS.

The following is an example of HTML code that opens the viewer in a new window:

```
<a  
href="http://s7dl.scene7.com/s7viewers/html5/VideoViewer.html?asset=Scene7SharedAssets/Glacier_Climber_MP4"  
target="_blank">Open popup viewer</a>
```

About fixed size embedding mode and responsive embedding mode

In the embedded mode, the viewer is added to the existing web page, which may already have some customer content not related to the viewer. The viewer normally occupies only a part of the web page's real estate.

The primary use case are web pages oriented for desktops or tablet devices, and also responsive design pages that adjust layout automatically depending on the device type.

Fixed size embedding is used when viewer does not change its size after initial load. This choice is the best for web pages that have a static page layout.

Responsive design embedding assumes that the viewer may need to resize in run-time in response to the size change of its container `DIV`. The most common use case is adding the viewer to a web page that uses a flexible page layout.

In responsive design embedding mode the viewer behaves differently depending on the way the web page sizes its container `DIV`. If the web page sets only the width of the container `DIV`, leaving its height unrestricted, the viewer automatically chooses its height according to the aspect ratio of the asset that is used. This method ensures that the asset fits perfectly into the view without any padding on the sides. This use case is the most common for web pages that use a responsive design layout framework like Bootstrap, Foundation, and so forth.

Otherwise, if the web page sets both the width and the height for the viewer's container `DIV`, the viewer fills just that area and follows the size provided by web page layout. A good example is embedding the viewer into a modal overlay, where the overlay is sized according to the size of the web browser window.

Fixed size embedding

You add the viewer to a web page by doing the following:

1. Adding the viewer JavaScript file to your web page.
2. Defining the container `DIV`.
3. Setting the viewer size.
4. Creating and initializing the viewer.

1. Adding the viewer JavaScript file to your web page.

Creating a viewer requires that you add a script tag in the HTML head. Before you can use the viewer API, be sure that you include `FlyoutViewer.js`. The `FlyoutViewer.js` file is located under the `html5/js/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/js/FlyoutViewer.js
```

You can use a relative path if the viewer is deployed on one of the Adobe Scene7 servers and it is served from the same domain. Otherwise, you specify a full path to one of Adobe Scene7 servers that have the IS-Viewers installed.

Relative path looks like the following:

```
<script language="javascript" type="text/javascript"
src="/s7viewers/html5/js/VideoViewer.js"></script>
```



Note: You should only reference the main viewer JavaScript include file on your page. You should not reference any additional JavaScript files in the web page code which might be downloaded by the viewer's logic in runtime. In particular, do not directly reference HTML5 SDK `Utils.js` library loaded by the viewer from `/s7viewers` context path (so-called consolidated SDK include). The reason is that the location of `Utils.js` or similar runtime viewer libraries is fully managed by the viewer's logic and the location changes between viewer releases. Adobe does not keep older versions of secondary viewer includes on the server.

As a result, putting a direct reference to any secondary JavaScript include used by the viewer on the page breaks the viewer functionality in the future when a new product version is deployed.

2. Defining the container DIV.

Add an empty DIV element to the page where you want the viewer to appear. The DIV element must have its ID defined because this ID is passed later to the viewer API. The DIV has its size specified through CSS.

The placeholder DIV is a positioned element, meaning that the `position` CSS property is set to `relative` or `absolute`.

Ensure that the full screen feature functions properly in Internet Explorer. Check to make sure that there are no other elements in the DOM that have a higher stacking order than your placeholder DIV.

The following is an example of a defined placeholder DIV element:

```
<div id="s7viewer" style="position:relative;width:640px;height:360px;"></div>
```

3. Setting the viewer size

You can set the static size for the viewer by either declaring it for `.s7videoviewer` top-level CSS class in absolute units, or by using the modifier `stagesize`.

Sizing in CSS can be put right on the HTML page, or in a custom viewer CSS file, which is later assigned to a viewer preset record in Scene7 Publishing System or passed explicitly using a style command.

See [Customizing Video Viewer](#) for more information about styling the viewer using CSS.

The following is an example of defining a static viewer size in an HTML page:

```
#s7viewer.s7videoviewer {
width: 640px;
height: 480px;
}
```

You can set `stagesize` modifier either in the viewer preset record in Scene7 Publishing System, or pass it explicitly with the viewer initialization code with `params` collection, or as an API call as described in the Command reference section, as in the following:

```
videoViewer.setParam("stagesize", "640,480");
```

A CSS-based approach is recommended and is used in this example.

4. Creating and initializing the viewer.

When you have completed the steps above, you create an instance of `s7viewers.VideoViewer` class, pass all configuration information to its constructor, and call `init()` method on a viewer instance. Configuration information is passed to the constructor as a JSON object. At minimum, this object should have `containerId` field which holds the name of viewer container ID and nested `params` JSON object with configuration parameters supported by the viewer. In this case, `params` object must have at least the Image Serving URL passed as `serverUrl` property, video server URL passed as `videoserverurl` property, and the initial asset as `asset` parameter. The JSON-based initialization API lets you create and start the viewer with a single line of code.

It is important to have the viewer container added to the DOM so that the viewer code can find the container element by its ID. Some browsers delay building DOM until the end of the web page. For maximum compatibility, call the `init()` method just before the closing `BODY` tag, or on the `body onload()` event.

At the same time, the container element should not necessarily be part of the web page layout just yet. For example, it may be hidden using `display:none` style assigned to it. In this case, the viewer delays its initialization process until the moment when the web page brings the container element back to the layout. When this occurs, the viewer load automatically resumes.

The following is an example of creating a viewer instance, passing minimum necessary configuration options to the constructor, and calling the `init()` method. This example assumes `videoViewer` is the viewer instance, `s7viewer` is the name of placeholder DIV, `http://s7d1.scene7.com/is/image/` is the Image Serving URL, `http://s7d1.scene7.com/is/content/` is the video server URL, and `Scene7SharedAssets/Glacier_Climber_MP4` is the asset.

```
<script type="text/javascript">
var videoViewer = new s7viewers.VideoViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/Glacier_Climber_MP4",
    "serverurl": "http://s7d1.scene7.com/is/image/",
    "videoserverurl": "http://s7d1.scene7.com/is/content/"
  }
}).init();
</script>
```

The following code is a complete example of a trivial web page that embeds the Video Viewer with a fixed size:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7d1.scene7.com/s7viewers/html5/js/VideoViewer.js"></script>
<style type="text/css">
#s7viewer.s7videoviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative;width:640px;height:360px;"></div>
<script type="text/javascript">
var videoViewer = new s7viewers.VideoViewer({
```



```

"containerId": "s7viewer",
"params": {
  "asset": "Scene7SharedAssets/Glacier_Climber_MP4",
  "serverurl": "http://s7dl.scene7.com/is/image/",
  "videoseverurl": "http://s7dl.scene7.com/is/content/"
}
}).init();
</script>
</body>
</html>

```

Responsive design embedding with unrestricted height

With the responsive design embedding, the web page normally has some kind of flexible layout in place that dictates the run-time size of the viewer's container DIV. For purposes of this example, assume that the web page allows the viewer's container DIV to take 40% of the web browser window size, leaving its height unrestricted. The web page HTML code would look like the following:

```

<!DOCTYPE html>
<html>
<head>
<style type="text/css">
.holder {
  width: 40%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>

```

Adding the viewer to such a page is very similar to the fixed size embedding; the only difference is that you do not need to explicitly define the viewer size.

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Creating and initializing the viewer.

All the steps above are the same as with the fixed size embedding. Add container DIV to the existing "holder" DIV. The following code is a complete example. You can see how the viewer size changes when the browser is resized, and how the viewer aspect ratio matches the asset.

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/VideoViewer.js"></script>
<style type="text/css">
.holder {
  width: 40%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var videoViewer = new s7viewers.VideoViewer({
  "containerId": "s7viewer",
"params": {
  "asset": "Scene7SharedAssets/Glacier_Climber_MP4",
  "serverurl": "http://s7dl.scene7.com/is/image/",
  "videoseverurl": "http://s7dl.scene7.com/is/content/"

```

```

}
}).init();
</script>
</body>
</html>

```

The following examples page illustrates more real-life use of responsive design embedding with unrestricted height:

https://marketing.adobe.com/resources/help/en_US/s7/vlist/vlist.html

Responsive design embedding with width and height defined

In case of responsive design embedding with width and height defined, the web page styling is different; it provides both sizes to the "holder" DIV and center it in the browser window. Also, the web page sets the size of the HTML and BODY element to 100%:

```

<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
height: 60%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>

```

The remaining embedding steps are identical to responsive design embedding with unrestricted height. The resulting example is the following:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/VideoViewer.js"></script>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
height: 60%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var videoViewer = new s7viewers.VideoViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Scene7SharedAssets/Glacier_Climber_MP4",

```

```

"serverurl": "http://s7dl.scene7.com/is/image/",
"videoserverurl": "http://s7dl.scene7.com/is/content/"
}
}).init();
</script>
</body>
</html>

```

Embedding using Setter-based API

Instead of using JSON-based initialization, it is possible to use setter-based API and no-args constructor. With that API, constructor does not take any parameters and configuration parameters are specified using `setContainerId()`, `setParam()`, and `setAsset()` API methods with separate JavaScript calls.

The following example illustrates fixed size embedding with setter-based API:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/VideoViewer.js"></script>
<style type="text/css">
#s7viewer.s7videoviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative;width:640px;height:360px;"></div>
<script type="text/javascript">
var videoViewer = new s7viewers.VideoViewer();
videoViewer.setContainerId("s7viewer");
videoViewer.setParam("serverurl", "http://s7dl.scene7.com/is/image/");
videoViewer.setParam("videoserverurl", "http://s7dl.scene7.com/is/content/");
videoViewer.setAsset("Scene7SharedAssets/Glacier_Climber_MP4");
videoViewer.init();
</script>
</body>
</html>

```

Command reference – Configuration attributes

Configuration attributes documentation for Video Viewer.

You can set any configuration command in the URL. Or, you can use the API methods `setParam()`, or `setParams()`, or both to set any configuration command. You can also specify any config attribute in the server-side configuration record.

You can prefix some configuration commands with the class name or the instance name of corresponding Viewer SDK component. An instance name of the component is dynamic and depends on the ID of the viewer container DOM element passed to `setContainerId()` API method. Documentation includes optional prefixes for such commands. For example, `playback` is documented as follows:

```
[VideoPlayer.|<containerId>_videoPlayer].playback
```

which means that this command is used in the following manner:

- `playback` (short syntax)
- `VideoPlayer.playback` (qualified with component class name)
- `cont_videoPlayer.playback` (qualified with component ID, assuming that `cont` is the ID of the container element)

See [Command reference common to all viewers – Configuration attributes](#)

See [Command reference common to all Viewers – URL](#)

ControlBar.transition

Configuration attribute for Video Viewer.

```
[ControlBar.|<containerId>_controls.]transition=none|fade[,delaytohide[,duration]]
```

<code>none fade</code>	<p>Specifies the effect type that is used to show or hide the control bar and its content.</p> <p>Use <code>none</code> for instant show and hide. Use <code>fade</code> to provide a gradual fade in and fade out effect.</p> <p>Fade is not supported on Internet Explorer 8.</p>
<code>delaytohide</code>	<p>Specifies the time in seconds between the last mouse/touch event that the control bar registers and the time control bar hides.</p> <p>If set to <code>-1</code> the component never triggers its auto-hide effect and always stays visible on the screen.</p>
<code>duration</code>	Sets the duration of the fade in and fade out animation, in seconds.

Properties

Optional.

Default

`fade,2,0.5`

Example

```
transition=none
```

EmailShare.emailurl

Configuration attribute for Video Viewer.

```
[EmailShare.|<containerId>_emailShare.]emailurl=emailurl
```

<code>emailurl</code>	Specifies the base URL for Scene7 OnDemand email service.
-----------------------	---

Properties

Optional.

Default

`/s7/emailFriend`

Example

```
emailurl=http://s7d1.scene7.com/s7/emailFriend
```

EmbedShare.embedsizes

Configuration attribute for Video Viewer.

[EmbedShare.|<containerId>_embedShare.]embedsizes=width,height[,0|1][;width,height[,0|1]]

Specifies a list of embed sizes for size combo box in the embed sharing modal dialog box.

width	Embed width.
height	Embed height.
0 1	Specifies whether this list item should be initially preselected in the combo box.

Properties

Optional.

Default

1280,960;640,480;320,240

Example

embedsizes=800,600;640,480,1

SocialShare.bearing

Configuration attribute for Video Viewer.

[SocialShare.|<containerId>_socialShare.]bearing=up|down|left|right|fit-vertical|fit-lateral

up down left right fit-vertical fit-lateral	<p>Specifies the direction of the slide animation for the buttons container.</p> <p>When set to up, down, left, or right, the panel rolls out in a specified direction without an additional bounds check, which can result in panel clipping by an outside container.</p> <p>When set to fit-vertical, the component first shifts the base panel position to the bottom of SocialShare and tries to roll out the panel from the bottom, right, or left from such base location. With each attempt the component checks if the panel is clipped by an outside container. If all attempts fail, the component tries to shift the base panel position to the top and repeat roll out attempts in the top, right, and left direction.</p> <p>When set to fit-lateral, the component uses a similar logic. However it shifts the base to the right first, trying right, down, and up rollout directions, and then shifts the base to the left, trying left, down, and up rollout directions.</p>
---	--

Properties

Optional.

Default

fit-vertical

Example

bearing=left

VideoPlayer.autoplay

Configuration attribute for Video Viewer.

[VideoPlayer.<containerId>_videoPlayer.]autoplay=0/1

0/1	Indicates whether the viewer starts playing the video on load. Some systems, like certain mobile devices, do not support AutoPlay.
-----	--

Properties

Optional.

Default

0

Example

autoplay=1

VideoPlayer.iconeffect

Configuration attribute for Video Viewer.

[VideoPlayer.<containerId>_videoPlayer.]iconeffect=0/1[,count][,fade][,autoHide]

0/1	Enables the IconEffect to display on top of the video when the video is paused. On some devices native controls are used. In such case, the iconeffect modifier is ignored.
count	Specifies the maximum number of times the IconEffect appears and reappears. A value of -1 indicates that the icon reappears indefinitely.
fade	Specifies the duration of show or hide animation, in seconds.
autoHide	Sets number of seconds that the IconEffect stays visible before it auto-hides. That is, the time after fade-in animation is completed and before fade-out animation starts. A setting of 0 disables auto-hide behavior.

Properties

Optional.

Default

1,-1,0.3,0

Example

```
iconeffect=0
```

VideoPlayer.initialbitrate

Configuration attribute for Video Viewer.

```
[VideoPlayer.<containerId>_videoPlayer.]initialbitrate=value
```

value	<p>Sets the video bitrate—in kbits per seconds or kbps—that is used for initial playback of video on desktops.</p> <p>If this bitrate value does not exist in the Adaptive Video Set, then the video player starts the video that has the next lowest bitrate.</p> <p>If set to 0 the video player starts from the lowest possible bitrate. Applicable only for systems that do not have native support for HTML5 HLS video (which are Firefox, Chrome and Internet Explorer 11 browsers on Windows 10), and for Flash-enabled desktop systems, and when playback mode is set to <code>auto</code>.</p>
-------	---

Properties

Optional.

Default

1400

Example

```
initialbitrate=600
```

VideoPlayer.loop

Configuration attribute for Video Viewer.

```
[VideoPlayer.<containerId>_videoPlayer.]loop=0|1
```

0 1	Indicates whether the video should play again after playback is complete.
-----	---

Properties

Optional.

Default

0

Example

```
loop=1
```

VideoPlayer.playback

Configuration attribute for Video Viewer.

```
[VideoPlayer.<containerId>_videoPlayer.] playback=auto|progressive|flash
```

auto progressive flash	<p>Sets the type of playback used by the viewer. When <code>auto</code> is set, on most desktop browsers and all iOS devices, the viewer uses HTML5 streaming video in HLS format. It falls back to progressive HTML5 playback on certain systems like older Internet Explorer and Android.</p> <p>If <code>progressive</code> is specified, the viewer relies only on HTML5 playback as natively supported by browsers and plays video progressively on all systems.</p> <p>If <code>flash</code> is specified, the viewer uses Flash video playback if the Flash Player is installed; otherwise, HTML5 playback is used.</p> <p>For more information on the playback selection in auto and progressive modes please consult the Viewer SDK User Guide.</p>
------------------------	--

Properties

Optional.

Ignored when viewer works with external video. See [External Video Support](#).

Default

auto

Example

```
playback=progressive
```

VideoPlayer.posterimage

Configuration attribute for Video Viewer.

```
[VideoPlayer.<containerId>_videoPlayer.] posterimage=none|[image_id][?isCommands]
```

none [image_id][?isCommands]	<p>The image to display on the first frame before the video starts playing, resolved against <code>serverurl</code>. If specified in the URL, HTTP-encode the following:</p> <ul style="list-style-type: none">• <code>?</code> as <code>%3F</code>• <code>&</code> as <code>%26</code>• <code>=</code> as <code>%3D</code> <p>If the <code>image_id</code> value is omitted, the component attempts to use the default poster image for that asset instead.</p>
------------------------------	--

When the video is specified as a path, the default poster images catalog id is derived from the video path as the `catalog_id/image_id` pair where `catalog_id` corresponds to the first token in the path and `image_id` is the name of the video with the extension removed. If the image with that ID does not exist, the poster image is not shown.

To prevent the display of the default poster image, specify `none` as the poster image value. If only the *isCommands* are specified the commands are applied to the default poster image before the image is displayed.

Properties

Optional.

Default

None.

Example

```
posterimage=none
```

VideoPlayer.progressivebitrate

Configuration attribute for Video Viewer.

```
[VideoPlayer.|<containerId>_videoPlayer.]progressivebitrate=value
```

value	<p>Specifies (in kbits per seconds or kbps) the desired video bit rate to play from an Adaptive Video Set in case the current system does not support adaptive video playback.</p> <p>The component picks up the video stream with the closest possible (but not exceeding) bitrate to the specified value. If all video streams in the Adaptive Video Set have higher quality than the specified value, the logic chooses the bitrate with the lowest quality.</p>
-------	---

Properties

Optional.

Default

700

Example

```
progressivebitrate=600
```

VideoPlayer.singleclick

Configuration attribute for Video Viewer.

```
[VideoPlayer.|<containerId>_videoPlayer.]singleclick= none/playPause
```

<code>none</code> <code>playPause</code>	Configures the mapping of single-click/tap to toggle play/pause. Setting to <code>none</code> disables single-click/tap to play/pause. If set to <code>playPause</code> , clicking the video toggles between playing and pausing the video. On some devices, you can use native controls. In such case, <code>singleclick</code> behavior is disabled.
--	--

Properties

Optional.

Default

`playPause`

Example

```
singleclick=none
```

VideoPlayer.smoothing

Configuration attribute for Video Viewer.

```
[VideoPlayer.<containerId>_videoPlayer.]smoothing=0|1
```

<code>0</code> <code>1</code>	<p>Specifies whether the video is smoothed (interpolated) when it is scaled. For smoothing to work, the runtime must be in high-quality mode (the default).</p> <p>The default value is <code>0</code> (no smoothing). Set this property to <code>1</code> to take advantage of mipmapping image optimization. Only for Flash playback.</p>
---------------------------------	---

Properties

Optional.

Default

`0`

Example

```
smoothing=1
```

VideoPlayer.ssl

Configuration attribute for Video Viewer.



Note: This configuration attribute only applies to AEM 6.2 with installation of [Feature Pack NPR-13480](#) and to AEM 6.1 with installation of [Feature Pack NPR-15011](#).

```
[VideoPlayer.<containerId>_videoPlayer.]ssl=auto|on
```

<code>auto</code> <code>on</code>	Controls whether the video is delivered over a secure SSL connection (HTTPS) or an insecure connection(HTTP).
-------------------------------------	---

	<p>When set to <code>auto</code> the video delivery protocol is inherited from the protocol of the embedding web page. If the web page is loaded over HTTPS, the video is also delivered over HTTPS, and vice versa. If the web page is on HTTP, the video is delivered over HTTP.</p> <p>When set to <code>on</code>, video delivery always occurs over a secure connection without regard to the web page protocol.</p> <p>Affects published video delivery only and is ignored for video preview in Author mode.</p>
--	---

Properties

Optional.

Default

`auto`

Example

```
ssl=on
```

See also [Secure Video Delivery](#).

VideoPlayer.waiticon

Configuration attribute for Video Viewer.

```
[VideoPlayer.<containerId>_videoPlayer.]waiticon=0|1
```

0 1	Enables or disables buffering animation (wait icon) display.
-----	--

Properties

Optional.

Default

1

Example

```
waiticon=0
```

VideoScrubber.chaptertimepattern

Configuration attribute for Video Viewer.

```
[VideoScrubber.<containerId>_videoScrubber.]chaptertimepattern=[h:]m|mm:s|ss
```

[h:]m mm:s ss	Sets the pattern for the time that is displayed in the title bar of the video chapter label, where <code>h</code> is hours, <code>m</code> is minutes, and <code>s</code> is seconds.
---------------	---

	<p>The number of letters used for each time unit determines the number of digits to display for the unit. If the number cannot fit into the given digits, the equivalent value is displayed in the subsequent unit.</p> <p>For example, if the current movie time is 67 minutes and 5 seconds, the time pattern <code>m:ss</code> shows as 67:05. The same time is displayed as 1:07:5 if the given time pattern is <code>h:mm:s</code>.</p>
--	--

Properties

Optional.

Default

`m:ss`

Example

```
chaptertimepattern=h:mm:ss
```

VideoScrubber.showchaptertime

Configuration attribute for Video Viewer.

```
[VideoScrubber.<containerId>_videoScrubber.]showchaptertime=0|1
```

0 1	Disables or enables the video chapter time in the title bar of the video chapter label.
-----	---

Properties

Optional.

Default

1

Example

```
showchaptertime=0
```

VideoScrubber.showchaptertitle

Configuration attribute for Video Viewer.

```
[VideoScrubber.<containerId>_videoScrubber.]showchaptertitle=0|1
```

0 1	Disables or enables the title bar of the video chapter label. The chapter start time is not displayed when the title bar is disabled.
-----	---

Properties

Optional.

Default

1

Example

```
showchaptertitle=0
```

VideoScrubber.timepattern

Configuration attribute for Video Viewer.

```
[VideoScrubber.<containerId>_videoScrubber.]timepattern=[h:]m|mm:s|ss
```

[h:]m mm:s ss	<p>Sets the pattern for the time that is displayed in the time bubble, where <i>h</i> is hours, <i>m</i> is minutes, and <i>s</i> is seconds.</p> <p>The number of letters used for each time unit determines the number of digits to display for the unit. If the number cannot fit into the given digits, the equivalent value is displayed in the subsequent unit.</p> <p>For example, if the current movie time is 67 minutes and 5 seconds, the time pattern <i>m:ss</i> shows as 67:05. The same time is displayed as 1:07:5 if the given time pattern is <i>h:mm:s</i>.</p>
---------------	--

Properties

Optional.

Default

m:ss

Example

```
timepattern=h:mm:ss
```

VideoTime.timepattern

Configuration attribute for Video Viewer.

```
[VideoTime.<containerId>_videoTime.]timepattern=[h:]m|mm:s|ss
```

[h:]m mm:s ss	<p>Sets the pattern for the time that is displayed in the control bar, where <i>h</i> is hours, <i>m</i> is minutes, and <i>s</i> is seconds.</p> <p>The number of letters used for each time unit determines the number of digits to display for the unit. If the number cannot fit into the given digits, the equivalent value is displayed in the subsequent unit.</p> <p>For example, if the current movie time is 67 minutes and 5 seconds, the time pattern <i>m:ss</i> shows as 67:05. The same time is displayed as 1:07:5 if the given time pattern is <i>h:mm:s</i>.</p>
---------------	--

Properties

Optional.

Default

m:ss

Example

```
timepattern=h:mm:ss
```

Command reference – URL

Command reference documentation Video Viewer.

You can set any configuration command in the URL. Or, you can use the API methods `setParam()`, or `setParams()`, or both to set any configuration command. You can also specify any config attribute in the server-side configuration record.

You can prefix some configuration commands with the class name or the instance name of corresponding Viewer SDK component. An instance name of the component is dynamic and depends on the ID of the viewer container DOM element passed to `setContainerId()` API method. Documentation includes optional prefixes for such commands. For example, `playback` is documented as follows:

```
[VideoPlayer.<containerId>_videoPlayer].playback
```

which means that this command is used in the following manner:

- `playback` (short syntax)
- `VideoPlayer.playback` (qualified with component class name)
- `cont_videoPlayer.playback` (qualified with component ID, assuming that `cont` is the ID of the container element)

See also [Command reference common to all viewers – Configuration attributes](#).

See also [Command reference common to all Viewers – URL](#).

caption

URL command for Video Viewer.

```
caption=file[,0|1]
```

The viewer supports closed captioning through hosted WebVTT files. Overlapping cues and regions are not supported. Supported cue positioning operators include the following:

Key	Name	Value	Description
A	text align	left right middle start end	Control text alignment. Default is middle.
T	text position	0%-100%	Percentage of inset into the VideoPlayer component for the beginning of the caption text. Default is 0%.

Key	Name	Value	Description
S	line size	0%-100%	Percentage of video width used for captions. Default is 100%.
L	line position	0%-100% integer	Determines the line position on the page. If it is expressed as an integer (no percent sign), then it is the number of lines from the top where the text is displayed. If it is a percentage (the percent sign is the last character), then the caption text is displayed that percent down the display area. Default is 100%.

Other WebVTT features present in the WebVTT file are not supported, but should not disrupt the captioning.

<i>file</i>	Specifies a URL or path to the WebVTT caption content. Serve the WebVTT file by ImageServing.
0 1	Specifies the default caption state (enabled is 1).

Properties

Optional.

Default

None.

Example

```
caption=Scene7SharedAssets/adobe_qbc_final_cc,1
```

navigation

URL command for Video Viewer.

```
navigation=file
```

The viewer supports video chapter navigation by way of hosted WebVTT files. Cue positioning operators are not supported.

<i>file</i>	Specifies a URL or path to WebVTT navigation content. Image Serving should host the WebVTT file.
-------------	--

Properties

Optional.

Default

None.

Example

```
navigation=Scene7SharedAssets/adobe_qbc_final_nc
```

video

URL command for Video Viewer.

```
video=videoURL
```

<i>videoURL</i>	The absolute URL to an external video.
-----------------	--

Properties

Required. (Unless `asset` parameter is used. See [External Video Support](#).)

Default

None.

Example

```
video=https://s7d9.scene7.com/is/content/Scene7SharedAssets/Glacier_Climber_MP4
```

videoServerUrl

URL command for Video Viewer.

```
videoServerUrl=videoRootPath
```

<i>videoRootPath</i>	The video server root path. If no domain is specified, then the domain from which the page is served is applied instead. Standard URI path resolution applies.
----------------------	--

Properties

Optional. Not needed for standard SaaS usage.

Default

```
/is/content/
```

Example

```
videoServerUrl=http://s7dl.scene7.com/is/content/
```

JavaScript API reference for Video Viewer

The main class of the Video Viewer is `VideoViewer`. It is declared in the `s7viewers` namespace. This JavaScript API covers constructor, methods, and callbacks of this particular class.

In all the following examples, `<instance>` is the actual name of the JavaScript viewer object that is instantiated from the `s7viewers.VideoViewer` class.

dispose

JavaScript API reference for Video Viewer.

```
dispose()
```

Disposes this viewer instance by releasing all resources used by the viewer logic and deleting all inner objects and components created by the viewer in runtime.

The web page code should also delete the viewer instance variable as well to completely remove the viewer from the web browser memory.

If the web page code has registered event listeners directly on Viewer SDK components used by the viewer—or stored external references to such components—such listeners must be explicitly unregistered by the web page code, and such external component references must be deleted prior to calling `dispose()`.

Do not access the Viewer API any more after `dispose()` is called.

Parameters

None.

Returns

None.

Example

```
<instance>.dispose()
```

getComponent

JavaScript API reference for Video Viewer

```
getComponent(componentId)
```

Returns a reference to the Viewer SDK component that is used by the viewer. The web page can use this method to extend or customize the behavior of the out-of-box viewer. Call this method only after the `initComplete` viewer callback has run, otherwise the component may not be created yet by the viewer logic.

Parameters

componentID – {String} an ID of the Viewer SDK component used by the viewer. This viewer supports the following component IDs:

Component ID	Viewer SDK component class name
parameterManager	s7sdk.ParameterManager
container	s7sdk.common.Container

Component ID	Viewer SDK component class name
mediaSet	s7sdk.set.MediaSet
videoPlayer	s7sdk.video.VideoPlayer
fullScreenButton	s7sdk.common.FullScreenButton
mutableVolume	s7sdk.video.MutableVolume
playPauseButton	s7sdk.common.PlayPauseButton
videoScrubber	s7sdk.video.VideoScrubber
videoTime	s7sdk.video.VideoTime
closedCaptionButton	s7sdk.common.ClosedCaptionButton
controls	s7sdk.common.ControlBar
socialShare	s7sdk.share.SocialShare
twitterShare	s7sdk.share.TwitterShare
facebookShare	s7sdk.share.FacebookShare
linkShare	s7sdk.share.LinkShare
emailShare	s7sdk.share.EmailShare
embedShare	s7sdk.share.EmbedShare

When working with SDK APIs it is important to use correct fully qualified SDK namespace as described in [Viewer SDK namespace](#).

Refer to the Viewer SDK API documentation for more information about a particular component.

Returns

{Object} a reference to Viewer SDK component. The method returns null if the componentId is not a supported viewer component or if the component was not yet created by the viewer logic.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    var videoPlayer = <instance>.getComponent("videoPlayer");
  }
})
```

init

JavaScript API reference for Video Viewer.

`init()`

Starts the initialization of the Video Viewer. By this time the container DOM element must be created so that the viewer code can find it by its ID.

If the container element is not a part of the web page layout just yet—for example, it may be hidden using `display:none` style assigned to it—the viewer suspends its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load automatically resumes.

Call this method only once during the viewer life cycle; subsequent calls are ignored.

Parameters

None.

Returns

{Object} A reference to the viewer instance.

Example

```
<instance>.init()
```

setAsset

JavaScript API reference for Video Viewer.

`setAsset(asset[, data])`

Sets the new asset and optional additional asset data. You can call this parameter at any time, either before or after `init()`. If it is called after `init()`, the viewer swaps the asset at runtime.

See also [init](#).

Parameters

asset	{String} new asset ID.
data	<p>{JSON} JSON object with the following optional fields (case-sensitive):</p> <ul style="list-style-type: none">• <code>posterimage</code> – Image to display on the first frame before the video starts playing. See VideoPlayer.posterimage.• <code>caption</code> – location of the new closed caption file. If the file is not specified, the closed caption button is not visible in the user interface.• <code>navigation</code> – URL or path to WebVTT navigation content. The WebVTT file should be served by Image Serving

Returns

None.

Example

```
<instance>.setAsset("Scene7SharedAssets/Glacier_Climber_MP4")
```

setContainerId

JavaScript API reference for Video Viewer.

```
setContainerId(containerId)
```

Sets the ID of the DOM container (normally a `DIV`) that the viewer is inserted into. It is not necessary to have the container element created by the time this method is called. However, the container must exist when `init()` is run. This parameter is called before `init()`. This method is optional if the viewer configuration information is passed with `config` JSON object to the constructor.

<i>containerId</i>	{string} ID of container.
--------------------	---------------------------

Returns

None.

Example

```
<instance>.setContainerId("s7viewer");
```

setHandlers

JavaScript API reference for Video Viewer.

```
setHandlers(handlers)
```

Specifies zero or more callback handlers. A call to this method fully overwrites event handlers that were previously assigned for that viewer instance. Must be called before `init()`.

Parameter

<i>handlers</i>	<p>{Object} JSON object with viewer event callbacks, where the property name is the name of the supported viewer event and the property value is a JavaScript function reference to an appropriate callback.</p> <p>See Event callbacks for more information about viewer events.</p>
-----------------	---

Returns

None.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    console.log("init complete");
  }
})
```

setLocalizedTexts

setLocalizedTexts(*localizationInfo*)

<i>localizationInfo</i>	<p>{Object} JSON object with localization data.</p> <p>See Viewer SDK namespace for more information.</p> <p>See the <i>Viewer SDK User Guide</i> and the example for more information about the object's content. Optional.</p>
-------------------------	--

Sets localization SYMBOL values for one or more locales. This parameter must be called before `init()`.

See also [init](#).

Returns

None.

Example

```
<instance>.setLocalizedTexts({ "en": { "VideoPlayer.ERROR": "Your Browser does not support HTML5 Video tag or the video cannot be played." }, "fr": { "VideoPlayer.ERROR": "Votre navigateur ne prend pas en charge la vidéo HTML5 tag ou la vidéo ne peuvent pas être lus." }, defaultLocale: "en" })
```

setParam

JavaScript API reference for Video Viewer.

setParam(*name*, *value*)

Sets the viewer parameter to a specified value. The parameter is either a viewer-specific configuration option or a software development kit modifier. This parameter is called before `init()`.

This method is optional if the viewer configuration information was passed with `config` JSON object to the constructor.

See also [init](#).

<i>name</i>	{string} name of parameter.
<i>value</i>	{string} value of parameter. The value cannot be percent-encoded.

Returns

None.

Example

```
<instance>.setParam("style", "customStyle.css")
```

setParams

JavaScript API reference for Video Viewer.

setParams(*params*)

Sets one or more parameters to a given value. The method argument syntax is identical to a URL query string. That is, it represents name=value pairs separated with &. The same as in a query string, names and values are percent-encoded using UTF8. Before you call `init()`, this parameter must be called.

This method is optional if the viewer configuration information is passed with `config` JSON object to the constructor.

See also [init](#).

<code>params</code>	<code>{string}</code> name=value parameter pairs separated with &.
---------------------	--

Returns

None.

Example

```
<instance>.setParams("style=customStyle.css&VideoPlayer.playback=progressive")
```

setVideo

JavaScript API reference for Video Viewer

```
setVideo(videoUrl[, data])
```

Sets new external video and optional additional video data. Can be called at any time, both before and after `init()`. If called after `init()`, the viewer swaps the video in run time.

See also [init](#).

Parameters

<code>videoUrl</code>	<code>{String}</code> an absolute URL to the new video.
<code>data</code>	<code>{JSON}</code> JSON object with the following optional fields (case-sensitive): <ul style="list-style-type: none">• <code>posterimage</code> – Image to display on the first frame before the video starts playing. See VideoPlayer.posterimage.• <code>caption</code> – Location of the new caption file. If no caption file is specified, the caption button is not displayed in the user interface.• <code>navigation</code> – URL or path to WebVTT navigation content. The WebVTT file should be served by Image Serving.

Returns

None.

Example

```
<instance>.setVideo("https://s7d9.scene7.com/is/content/Scene7SharedAssets/Glacier_Climber_MP4")
```

VideoViewer

JavaScript API reference for Video Viewer.

```
VideoViewer([config])
```

Constructor; creates a new Video Viewer instance.

Parameters

<i>config</i>	<p>{Object} optional JSON configuration object, allows all the viewer settings to pass to the constructor and avoid calling individual setter methods. Contains the following properties:</p> <ul style="list-style-type: none"> • containerId – {String} ID of the DOM container (normally a <code>DIV</code>) that the viewer is inserted into. It is not necessary to have the container element created by the time this method is called. However, the container must exist when <code>init()</code> is run. Required. • params – {Object} JSON object with viewer configuration parameters where the property name is either a viewer-specific configuration option or an SDK modifier, and the value of that property is a corresponding settings value. Required. • handlers – {Object} JSON object with viewer event callbacks, where the property name is the name of supported viewer event, and the property value is a JavaScript function reference to an appropriate callback. Optional. See Event callbacks for more information about viewer events. • localizedTexts – {Object} JSON object with localization data. Optional. See Viewer SDK namespace for more information. See the <i>Viewer SDK User Guide</i> and the example for more information about the object's content. Optional.
---------------	---

Returns

None.

Example

```
var videoViewer = new s7viewers.VideoViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/Glacier_Climber_MP4",
    "serverurl": "http://s7dl.scene7.com/is/image/",
    "videoserverurl": "http://s7dl.scene7.com/is/content/"
  },
  "handlers": {
    "initComplete": function() {
      console.log("init complete");
    }
  },
  "localizedTexts": {
    "en": {
      "VideoPlayer.ERROR": "Your Browser does not support HTML5 Video tag or the video cannot be played."
    },
    "fr": {
      "VideoPlayer.ERROR": "Votre navigateur ne prend pas en charge la vidéo HTML5 tag ou la vidéo ne peuvent pas être lus."
    }
  },
  defaultLocale: "en"
});
```

Event callbacks

The viewer supports JavaScript event callbacks that the web page uses to track the viewer initialization process or runtime behavior.

Callback handlers are assigned by passing event names and corresponding handler functions with the `handlers` property to `config` JSON object in the viewer's constructor. Alternatively, it is possible to use `setHandlers()` API method.

Supported viewer events include the following:

- `initComplete` – triggers when viewer initialization is complete and all internal components are created, so that it is possible to use `getComponent()` API. The callback handler does not take any arguments.
- `trackEvent` – triggers each time an event occurs inside the viewer which may be handled by an event tracking system, such as Adobe Analytics. The callback handler takes the following arguments:
 - `objID` {String} not currently used.
 - `compClass` {String} not currently used.
 - `instName` {String} an instance name of the Viewer SDK component that triggered the event.
 - `timeStamp` {Number} event time stamp.
 - `eventInfo` {String} event payload.

See also [VideoViewer](#) and [setHandlers](#).

Customizing Video Viewer

All visual customization and most behavior customization is done by creating a custom CSS.

The suggested workflow is to take the default CSS file for the appropriate viewer, copy it to a different location, customize it, and specify the location of the customized file in the `style=` command.

Default CSS files can be found at the following location:

```
<s7_viewers_root>/html5/VideoViewer.css
```

Custom CSS file must contain the same class declarations as the default one. If a class declaration is omitted, the viewer does not function properly because it does not provide built-in default styles for user interface elements.

An alternative way to provide custom CSS rules is to use embedded styles directly on the web page or in one of linked external CSS rules.

When you create custom CSS remember that the viewer assigns `.s7videoviewer` class to its container DOM element. If you use an external CSS file that is passed with the `style=` command, use `.s7videoviewer` class as parent class in descendant selector for your CSS rules. If you embed styles on the web page, additionally qualify this selector with an ID of the container DOM element as follows:

```
#<containerId>.s7videoviewer
```

Building responsive designed CSS

It is possible to target different devices in CSS to make your content display differently depending on a user's device. This targeting includes, but is not limited to, different user interface element sizes and artwork resolution.

The viewer supports two mechanisms of creating responsive designed CSS: CSS markers and standard CSS media queries. You can use these independently or together.

CSS markers

To assist in creating responsive designed CSS, the viewer supports CSS markers which special CSS classes dynamically assigned to the top-level viewer container element based on the run-time viewer size and the input type used on the current device.

The first group of CSS markers includes `.s7size_large`, `.s7size_medium`, and `.s7size_small` classes. They are applied based on the runtime area of the viewer container. That is, if the viewer area is equal to or bigger than the size of a common desktop monitor `.s7size_large` is used; if the area is close in size to a common tablet device `.s7size_medium` is assigned. For areas similar to mobile phone screens `.s7size_small` is set. The primary purpose of these CSS markers is to create different user interface layouts for different screens and viewer sizes.

The second group of CSS markers includes `.s7mouseinput` and `.s7touchinput`. `.s7touchinput` is set if the current device has touch input capabilities; otherwise, `.s7mouseinput` is used. These markers are intended to create user interface input elements with different screen sizes for different input types, because normally touch input requires larger elements. In case the device has both mouse input and touch capabilities, `.s7touchinput` is set and the viewer renders a touch-friendly user interface.

The following sample CSS sets the play/pause button size to 28 x 28 pixels on systems with mouse input, and 56 x 56 pixels on touch devices. In addition, it hides the button completely if the viewer size becomes really small:

```
.s7videoviewer.s7mouseinput .s7playpausebutton {
    width:28px;
    height:28px;
}
.s7videoviewer.s7touchinput .s7playpausebutton {
    width:56px;
    height:56px;
}
.s7videoviewer.s7size_small .s7playpausebutton {
    visibility:hidden;
}
```

To target devices with a different pixel density, use CSS media queries. The following media query block would contain CSS that is specific to high density screens:

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5)
{
}
```

Using CSS markers is the most flexible way of building responsive designed CSS as it allows you to target not only device screen size but actual viewer size, which may be useful for responsive design page layouts.

Use the default viewer CSS file as an example of a CSS markers approach.

CSS media queries

Device sensing can also be done using pure CSS media queries. Everything enclosed within a given media query block is applied only when it is run on a corresponding device.

When applied to Mobile Viewers, use four CSS media queries, defined in your CSS in the order listed below:

1. Contains only rules specific for all touch devices.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px)
{
}
```

2. Contains only rules specific for tablets with high resolution screens.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px) and (-webkit-min-device-pixel-ratio:1.5),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px) and (-webkit-min-device-pixel-ratio:1.5)
{
}
```

3. Contains only rules specific for all mobile phones.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in)
{
}
```

4. Contains only rules specific for mobile phones with high resolution screens.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in) and
(-webkit-min-device-pixel-ratio: 1.5),
only screen and (device-width:720px) and (device-height:1280px) and
(-webkit-device-pixel-ratio: 2),
only screen and (device-width:1280px) and (device-height:720px) and
(-webkit-device-pixel-ratio: 2)
{
}
```

Using the CSS media queries approach, you should organize CSS with device sensing as follows:

- First, the desktop-specific section defines all properties which are either desktop-specific or common to all screens.
- Second, the four media queries should go in order defined above and provide CSS rules specific for corresponding device type.

There is no need to duplicate the entire viewer CSS in each media query. Only properties that are specific to given devices are redefined inside a media query.

CSS Sprites

Many viewer user interface elements are styled using bitmap artwork and have more than one distinct visual state. A good example is a button that normally has at least 3 different states: "up", "over", and "down". Each state requires its own bitmap artwork assigned.

With a classic approach to styling, the CSS would have a separate reference to individual image file on the server for each state of the user interface element. The following is a sample CSS for styling a full screen button:

```
.s7videoviewer.s7mouseinput .s7playpausebutton[selected='true'][state='up'] {
background-image:url(images/v2/PlayButton_up.png);
}
.s7videoviewer.s7mouseinput .s7playpausebutton[selected='true'][state='over'] {
background-image:url(images/v2/PlayButton_over.png);
}
.s7videoviewer.s7mouseinput .s7playpausebutton[selected='true'][state='down'] {
background-image:url(images/v2/PlayButton_down.png);
}
.s7videoviewer.s7mouseinput .s7playpausebutton[selected='true'][state='disabled'] {
background-image:url(images/v2/PlayButton_disabled.png);
}
.s7videoviewer.s7mouseinput .s7playpausebutton[selected='false'][state='up'] {
background-image:url(images/v2/PauseButton_up.png);
}
.s7videoviewer.s7mouseinput .s7playpausebutton[selected='false'][state='over'] {
background-image:url(images/v2/PauseButton_over.png);
}
.s7videoviewer.s7mouseinput .s7playpausebutton[selected='false'][state='down'] {
background-image:url(images/v2/PauseButton_down.png);
}
}
```

```
.s7videoviewer.s7mouseinput .s7playpausebutton[selected='false'][state='disabled'] {
background-image:url(images/v2/PauseButton_disabled.png);
}
.s7videoviewer.s7mouseinput .s7playpausebutton[selected='true'][replay='true'][state='up'] {
background-image:url(images/v2/ReplayButton_up.png);
}
.s7videoviewer.s7mouseinput .s7playpausebutton[selected='true'][replay='true'][state='over'] {
background-image:url(images/v2/ReplayButton_over.png);
}
.s7videoviewer.s7mouseinput .s7playpausebutton[selected='true'][replay='true'][state='down'] {
background-image:url(images/v2/ReplayButton_down.png);
}
.s7videoviewer.s7mouseinput .s7playpausebutton[selected='true'][replay='true'][state='disabled'] {
background-image:url(images/v2/ReplayButton_disabled.png);
}
```

The drawback to this approach is that the end user experiences flickering or delayed user interface response when the element is interacted with for the first time. This action occurs because the image artwork for the new element state is not yet downloaded. Also, this approach may have a slight negative impact on performance because of an increase in the number of HTTP calls to the server.

CSS sprites is a different approach where image artwork for all element states is combined into a single PNG file called a "sprite". Such "sprite" has all visual states for the given element positioned one after another. When styling a user interface element with sprites the same sprite image is referenced for all different states in the CSS. Also, the `background-position` property is used for each state to specify which part of the "sprite" image is used. You can structure a "sprite" image in any suitable way. Viewers normally have it vertically stacked. Below is a "sprite"-based example of styling the same full screen button from above:

```
.s7videoviewer .s7fullscreenbutton[state][selected]{
background-image: url(images/v2/FullScreenButton_dark_sprite.png);
}
.s7videoviewer.s7mouseinput .s7fullscreenbutton[selected='true'][state='up'] {
background-position: -84px -1148px;
}
.s7videoviewer.s7mouseinput .s7fullscreenbutton[selected='true'][state='over'] {
background-position: -56px -1148px;
}
.s7videoviewer.s7mouseinput .s7fullscreenbutton[selected='true'][state='down'] {
background-position: -28px -1148px;
}
.s7videoviewer.s7mouseinput .s7fullscreenbutton[selected='true'][state='disabled'] {
background-position: -0px -1148px;
}
.s7videoviewer.s7mouseinput .s7fullscreenbutton[selected='false'][state='up'] {
background-position: -84px -1120px;
}
.s7videoviewer.s7mouseinput .s7fullscreenbutton[selected='false'][state='over'] {
background-position: -56px -1120px;
}
.s7videoviewer.s7mouseinput .s7fullscreenbutton[selected='false'][state='down'] {
background-position: -28px -1120px;
}
.s7videoviewer.s7mouseinput .s7fullscreenbutton[selected='false'][state='disabled'] {
background-position: -0px -1120px;
}
```

General styling notes and advice

- All paths to external assets within CSS are resolved against the CSS location not the location of the viewer HTML page. Remember to take this rule into account when you copy the default CSS to a different location. Copy the default assets or update paths within the custom CSS.
- The preferred format for bitmap artwork is PNG.

- Bitmap artwork is assigned to user interface elements using the `background-image` property.
- The `width` and `height` properties of a user interface element define its logical size. The size of the bitmap passed to `background-image` does not affect logical size.
- To use the high pixel density of high-resolution screens like Retina, specify bitmap artwork twice as large as the logical user interface element size. Then, apply the `-webkit-background-size:contain` property to scale the background down to the logical user interface element size.
- To remove a button from the user interface, add `display:none` to its CSS class.
- You can use various formats for color value that CSS supports. If you need transparency, use the format `rgba(R,G,B,A)`. Otherwise, you can use the format `#RRGGBB`.
- When customizing the viewer user interface with CSS the use of the `!important` rule is not supported to style viewer elements. In particular, `!important` rule should not be used to override any default or run-time styling provided by the viewer or Viewer SDK. The reason is that it may affect the behavior of proper components. Instead, you should use CSS selectors with the proper specificity to set CSS properties that are documented in this reference guide.

Common User Interface Elements

The following is user interface elements reference documentation that applies to Video Viewer:

Caption button

This button toggles closed caption display on and off. It is not visible if the caption parameter is not specified.


You can size, skin, and position this button, relative to the control bar that contains it, by using CSS.

The appearance of this button is controlled with the following CSS class selector:

```
.s7videoviewer .s7closedcaptionbutton
```

CSS properties of the caption button

<code>top</code>	Position from the top border, including padding.
<code>right</code>	Position from the right border, including padding.
<code>left</code>	Position from the left border, including padding.
<code>bottom</code>	Position from the bottom border, including padding.
<code>width</code>	The width of the full screen button.
<code>height</code>	The height of the full screen button.
<code>background-image</code>	The displayed image for a given button state.
<code>background-position</code>	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

 **Note:** This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the state when captions are visible and *selected='false'* is used when captions are hidden.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example

To set up a closed caption button that is 28 x 28 pixels, positioned four pixels from the top and 68 pixels from the right edge of the control bar, and displays a different image for each of the four different button states when selected or not selected.

```
.s7videoviewer .s7closedcaptionbutton {
  position:absolute;
  top:4px;
  right:68px;
  width:28px;
  height:28px;
}
.s7videoviewer .s7closedcaptionbutton[selected='true'][state='up'] {
background-image:url(images/v2/ClosedCaptionButton_up.png);
}
.s7videoviewer .s7closedcaptionbutton[selected='true'][state='over'] {
background-image:url(images/v2/ClosedCaptionButton_over.png);
}
.s7videoviewer .s7closedcaptionbutton[selected='true'][state='down'] {
background-image:url(images/v2/ClosedCaptionButton_down.png);
}
.s7videoviewer .s7closedcaptionbutton[selected='true'][state='disabled'] {
background-image:url(images/v2/ClosedCaptionButton_disabled.png);
}
.s7videoviewer .s7closedcaptionbutton[selected='false'][state='up'] {
background-image:url(images/v2/ClosedCaptionButton_disabled.png);
}
.s7videoviewer .s7closedcaptionbutton[selected='false'][state='over'] {
background-image:url(images/v2/ClosedCaptionButton_over.png);
}
.s7videoviewer .s7closedcaptionbutton[selected='false'][state='down'] {
background-image:url(images/v2/ClosedCaptionButton_down.png);
}
.s7videoviewer .s7closedcaptionbutton[selected='false'][state='disabled'] {
background-image:url(images/v2/ClosedCaptionButton_disabled.png);
}
```

Control bar

The control bar is the rectangular area that contains and sits behind all the UI controls available for the video viewer, such as the play/pause button, volume controls, and so on.

The control bar always takes the entire available viewer width. It is possible to change its color, height, and vertical position by CSS, relative to the video viewer container.

The following CSS class selector controls the appearance of the control bar:

```
.s7videoviewer .s7controlbar
```

CSS properties of the control bar

top	Position from the top border, including padding.
bottom	Position from the bottom border, including padding.

height	Height of the control bar.
background-color	Background color of the control bar.

Example

To set up a video viewer with a gray control bar that is 30 pixels tall and sits at the top of the video viewer container.

```
.s7videoviewer .s7controlbar {
position: absolute;
top: 0px;
height: 30px;
background-color: rgb(51, 51, 51);
}
```

Email share

Email share tool consists of a button added to the Social share panel and the modal dialog box which displays when the tool is activated. The position of the button is fully managed by the Social share tool.

The appearance of the email share button is controlled with the following CSS class selector:

```
.s7videoviewer .s7emailshare
```

CSS properties of the email share tool

width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a email share button that is 28 x 28 pixels, and that displays a different image for each of the four different button states.

```
.s7videoviewer .s7emailshare {
width:28px;
height:28px;
}
.s7videoviewer .s7emailshare[state='up'] {
background-image:url(images/v2/EmailShare_dark_up.png);
}
.s7videoviewer .s7emailshare[state='over'] {
background-image:url(images/v2/EmailShare_dark_over.png);
}
.s7videoviewer .s7emailshare[state='down'] {
background-image:url(images/v2/EmailShare_dark_down.png);
}
```

```
.s7videoviewer .s7emailshare[state='disabled'] {
background-image:url(images/v2/EmailShare_dark_disabled.png);
}
```

The background overlay which covers web page when the dialog is active is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7backoverlay
```

CSS properties of the back overlay

opacity	Background overlay opacity.
background-color	Background overlay color.

Example – to set up background overlay to be gray with 70% opacity:

```
.s7videoviewer .s7emaildialog .s7backoverlay {
  opacity:0.7;
  background-color:#222222;
}
```

By default the modal dialog is displayed centered in the screen on desktop systems and takes the whole web page area on touch devices. In all cases, the positioning and sizing of the dialog box is managed by the component. The dialog is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialog
```

CSS properties of the dialog box

border-radius	Dialog box border radius (in case the dialog box does not take the entire browser window);
background-color	Dialog box background color;
width	Should be either unset, or set to 100%, in which case the dialog takes the whole browser window (this mode is preferred on touch devices);
height	Should be either unset, or set to 100%, in which case the dialog takes the whole browser window (this mode is preferred on touch devices).

Example – to set up dialog to use entire browser window and have white background on touch devices:

```
.s7videoviewer .s7touchinput .s7emaildialog .s7dialog {
  width:100%;
  height:100%;
  background-color: #ffffff;
}
```

The dialog box header consists of an icon, a title text and a close button. The header container is controlled with the following CSS class selector

```
.s7videoviewer .s7emaildialog .s7dialogheader
```

CSS properties of the dialog box header

padding	Inner padding for header content.
---------	-----------------------------------

The icon and the title text are wrapped into an additional container controlled with

```
.s7videoviewer .s7emaildialog .s7dialogheader .s7dialogline
```

CSS properties of the dialog line

padding	Inner padding for the header icon and title
---------	---

Header icon is controlled with the following CSS class selector

```
.s7videoviewer .s7emaildialog .s7dialogheadericon
```

CSS properties of the dialog box header icon

width	Icon width.
height	Icon height.
background-image	Icon image.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

Header title is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogheadertext
```

CSS properties of the dialog box header text

font-weight	Font weight.
font-size	Font height.
font-family	Font family.
padding	Internal text padding.


Close button is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7closebutton
```

CSS properties of the close button

top	Vertical button position relative to header container.
right	Horizontal button position relative to header container.
width	Button width.
height	Button height.

padding	Inner padding of button.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

 **Note:** This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The Close button tool tip and the dialog box title can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up dialog header with padding, 24 x 17 pixels icon, bold 16 pt title and 28 x 28 pixels close button positioned 2 pixels from the top and 2 pixels from the right of dialog container:

```
.s7videoviewer .s7emaildialog .s7dialogheader {
padding: 10px;
}
.s7videoviewer .s7emaildialog .s7dialogheader .s7dialogline {
padding: 10px 10px 2px;
}
.s7videoviewer .s7emaildialog .s7dialogheadericon {
background-image: url("images/sdk/dlgemail_cap.png");
height: 17px;
width: 24px;
}
.s7videoviewer .s7emaildialog .s7dialogheadertext {
font-size: 16pt;
font-weight: bold;
padding-left: 16px;
}
.s7videoviewer .s7emaildialog .s7closebutton {
top:2px;
right:2px;
padding:8px;
width:28px;
height:28px;
}
.s7videoviewer .s7emaildialog .s7closebutton[state='up'] {
background-image:url(images/sdk/close_up.png);
}
.s7videoviewer .s7emaildialog .s7closebutton[state='over'] {
background-image:url(images/sdk/close_over.png);
}
.s7videoviewer .s7emaildialog .s7closebutton[state='down'] {
background-image:url(images/sdk/close_down.png);
}
.s7videoviewer .s7emaildialog .s7closebutton[state='disabled'] {
background-image:url(images/sdk/close_disabled.png);
}
```

Dialog footer consists of "cancel" and "send email" buttons. The footer container is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogfooter
```

CSS properties of the dialog box footer

border	Border which may be used to visually separate the footer from the rest of the dialog box.
--------	---

The footer has an inner container which keeps both buttons. It is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogbuttoncontainer
```

CSS properties of the dialog box button container

padding	Inner padding between the footer and the buttons.
---------	---

Cancel button is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogcancelbutton
```

CSS properties of the dialog box cancel button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

Send email button is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogactionbutton
```

CSS properties of the dialog box action button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

In addition, both buttons share the same common CSS class which can contain CSS settings that are the same for other dialog box buttons:

```
.s7videoviewer .s7emaildialog .s7dialogfooter .s7button
```

CSS properties of the button

font-weight	Button font weight.
font-size	Button font size.

font-family	Button font family.
line-height	Text height inside the button. Affects vertical alignment.
box-shadow	Drop shadow.
margin-right	Right button margin.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box footer with 64 x 34 Cancel button and a 82 x 34 send email button, with the text color and background color different for each button state:

```
.s7videoviewer .s7emaildialog .s7dialogfooter {
  border-top: 1px solid #909090;
}
.s7videoviewer .s7emaildialog .s7dialogbuttoncontainer {
  padding-bottom: 6px;
  padding-top: 10px;
}
.s7videoviewer .s7emaildialog .s7dialogfooter .s7button {
  box-shadow: 1px 1px 1px #999999;
  color: #FFFFFF;
  font-size: 9pt;
  font-weight: bold;
  line-height: 34px;
  margin-right: 10px;
}
.s7videoviewer .s7emaildialog .s7dialogcancelbutton {
  width: 64px;
  height: 34px;
}
.s7videoviewer .s7emaildialog .s7dialogcancelbutton[state='up'] {
  background-color: #666666;
  color: #dddddd;
}
.s7videoviewer .s7emaildialog .s7dialogcancelbutton[state='down'] {
  background-color: #555555;
  color: #ffffff;
}
.s7videoviewer .s7emaildialog .s7dialogcancelbutton[state='over'] {
  background-color: #555555;
  color: #ffffff;
}
.s7videoviewer .s7emaildialog .s7dialogcancelbutton[state='disabled'] {
  background-color: #b2b2b2;
  color: #dddddd;
}
.s7videoviewer .s7emaildialog .s7dialogactionbutton {
  width: 82px;
  height: 34px;
}
.s7videoviewer .s7emaildialog .s7dialogactionbutton[state='up'] {
  background-color: #333333;
  color: #dddddd;
}
.s7videoviewer .s7emaildialog .s7dialogactionbutton[state='down'] {
  background-color: #222222;
  color: #cccccc;
}
.s7videoviewer .s7emaildialog .s7dialogactionbutton[state='over'] {
  background-color: #222222;
  color: #cccccc;
}
```

```

}
.s7videoviewer .s7emaildialog .s7dialogactionbutton[state='disabled'] {
  background-color:#b2b2b2;
  color:#dddddd;
}

```

The main dialog area (between the header and the footer) contains scrollable dialog content and scroll panel on the right. In all cases, the component manages the width of this area, it is not possible to set it in CSS. Main dialog area is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogviewarea
```

CSS properties of the dialog box viewing area

height	The height of the main dialog box area. It should be specified only when the dialog box works in desktop mode. It is not applicable when the dialog box is sized to occupy the entire browser window.
background-color	The background color of the main dialog box area.
margin	Outer margin.



Note: The main dialog box area supports the optional *state* attribute selector. It is set to *sendsuccess* when the email form is submitted and the dialog box shows a confirmation message. As long as the confirmation message is small, this attribute selector can be used to reduce the dialog box height when such confirmation message is displayed.

Example – to set up the main dialog box area to be 300 pixels height initially and 100 pixels height when the confirmation message is shown, have a ten pixel margin, and use a white background:

```

.s7videoviewer .s7emaildialog .s7dialogviewarea {
  background-color:#ffffff;
  margin:10px;
  height:300px;
}
.s7videoviewer .s7emaildialog .s7dialogviewarea[state='sendsuccess'] {
  height:100px;
}

```

All form content (like labels and input fields) resides inside a container controlled with

```
.s7videoviewer .s7emaildialog .s7dialogbody
```

If the height of this container appears to be bigger than the main dialog box area, a vertical scroll is enabled automatically by the component.

CSS properties of the dialog box body

padding	Inner padding.
---------	----------------

Example – to set up form content to have ten pixel padding:

```

.s7videoviewer .s7emaildialog .s7dialogbody {
  padding: 10px;
}

```

Dialog box form is filled on line-by-line basis, where each line carries a part of the form content (like a label and a text input field). Single form line is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogbody .s7dialogline
```

CSS properties of the dialog box line

padding	Inner line padding.
---------	---------------------

Example – to set up a dialog box form to have ten pixel padding for each line:

```
.s7videoviewer .s7emaildialog .s7dialogbody .s7dialogline {
    padding: 10px;
}
```

All static labels in the dialog box form are controlled with

```
.s7videoviewer .s7emaildialog .s7dialoglabel
```

This class is not suitable for controlling labels size or position because you can apply it to texts in various places of the form user interface.

CSS properties of the dialog box label.

font-weight	Label font weight.
font-size	Label font size.
font-family	Label font family.
color	Label text color.

The dialog box labels can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up all labels to be gray, bold with a nine pixel font:

```
.s7videoviewer .s7emaildialog .s7dialoglabel {
    color: #666666;
    font-size: 9pt;
    font-weight: bold;
}
```

All static labels that are displayed to the left of the form input fields are controlled with:

```
.s7videoviewer .s7emaildialog .s7dialoginputlabel
```

CSS properties of the dialog box input label

width	The width of the static label.
text-align	The horizontal text alignment.
margin	Static label margin.
padding	Static label padding.

Example – to set up input field labels to be 50 pixels width, right-aligned, have ten pixels of padding, and a ten pixel margin on the right:

```
.s7videoviewer .s7emaildialog .s7dialoginputlabel {
  margin-right: 10px;
  padding: 10px;
  text-align: right;
  width: 50px;
}
```

Each form input field is wrapped into the container which lets you apply a custom border around the input field. It is controlled with the following CSS class selector::

```
.s7videoviewer .s7emaildialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

border	Border around the input field container.
padding	Inner padding.



Note: Input field container supports optional *state* attribute selector. It is set to *verifyerror* when the user makes a mistake in input data format and inline validation fails. This attribute selector can be used to highlight incorrect user input in the form.

Most input fields that spread from the label on the left up to the right edge of the dialog box body (which includes "from" field and "message" field) are controlled with:

```
.s7videoviewer .s7emaildialog .s7dialoginputwide
```

CSS properties of the dialog box input wide field

width	Input field width.
-------	--------------------

"To" input field is narrower because it allocates space for "remove email" button on the right. It is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialoginputshort
```

CSS properties of the dialog box input short field

width	Input field width.
-------	--------------------

Example – to set up a form to have a one pixel grey border with nine pixels of padding around all input fields; to have the same border in red color for fields which fail validation, to have 250 pixels wide "To" input field, and the rest of the input fields 300 pixels wide:

```
.s7videoviewer .s7emaildialog .s7dialoginputcontainer {
  border: 1px solid #CCCCCC;
  padding: 9px;
}
.s7videoviewer .s7emaildialog .s7dialoginputcontainer[state="verifyerror"] {
  border: 1px solid #FF0000;
}
.s7videoviewer .s7emaildialog .s7dialoginputshort {
  width: 250px;
}
```

```
.s7videoviewer .s7emaildialog .s7dialoginputwide {
  width: 300px;
}
```

Email message input field is additionally controlled with:

```
.s7videoviewer .s7emaildialog .s7dialogmessage
```

This class lets you set specific properties for the underlying TEXTAREA element.

CSS properties of the dialog box message

height	Message height.
word-wrap	Word wrapping style.

Example – to set up an email message to be 50 pixels high and use break-word word wrapping:

```
.s7videoviewer .s7emaildialog .s7dialogmessage {
  height: 50px;
  word-wrap: break-word;
}
```

"Add Another Email Address" button allows a user to add more than one addressee in email form. It is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogaddemailbutton
```

CSS properties of the dialog box add email address button

height	Button height.
color	Button text color for each state.
background-image	Button image for each state.
background-position	Button image position inside the button area.
font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.
line-height	Text height inside the button. Affects the vertical alignment.
text-align	Horizontal text alignment.
padding	Inner padding.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up "Add Another Email Address" button to be 25 pixels high, use 12 point bold font with right alignment, and a different text color and image for each state:

```
.s7videoviewer .s7emaildialog .s7dialogaddemailbutton {
  text-align:right;
  font-size:12pt;
  font-weight:bold;
  background-position:left center;
  line-height:25px;
  padding-left:30px;
  height:25px;
}
.s7videoviewer .s7emaildialog .s7dialogaddemailbutton[state='up'] {
  color:#666666;
  background-image:url(images/sdk/dlgaddplus_up.png);
}
.s7videoviewer .s7emaildialog .s7dialogaddemailbutton[state='down'] {
  color:#000000;
  background-image:url(images/sdk/dlgaddplus_over.png);
}
.s7videoviewer .s7emaildialog .s7dialogaddemailbutton[state='over'] {
  color:#000000;
  text-decoration:underline;
  background-image:url(images/sdk/dlgaddplus_over.png);
}
.s7videoviewer .s7emaildialog .s7dialogaddemailbutton[state='disabled'] {
  color:#666666;
  background-image:url(images/sdk/dlgaddplus_up.png);
}
```

"Remove" button lets a user remove extra addressees from the email form. It is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogremoveemailbutton
```

CSS properties of the dialog box remove email button

width	Button width.
height	Button height.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a "Remove" button to be 25 x 25 pixels and use a different image for each state:

```
.s7videoviewer .s7emaildialog .s7dialogremoveemailbutton {
  width:25px;
  height:25px;
}
.s7videoviewer .s7emaildialog .s7dialogremoveemailbutton[state='up'] {
  background-image:url(images/sdk/dlgremove_up.png);
}
.s7videoviewer .s7emaildialog .s7dialogremoveemailbutton[state='over'] {
```



```
background-image:url(images/sdk/dlgremove_over.png);
}
.s7videoviewer .s7emaildialog .s7dialogremoveemailbutton[state='down'] {
background-image:url(images/sdk/dlgremove_over.png);
}
.s7videoviewer .s7emaildialog .s7dialogremoveemailbutton[state='disabled'] {
background-image:url(images/sdk/dlgremove_up.png);
}
```

The content being shared is displayed in the bottom of the dialog box body and includes a thumbnail, title, origin URL, and description. It is wrapped into container controlled with:

```
.s7videoviewer .s7emaildialog .s7dialogbody .s7dialogcontent
```

CSS properties of the dialog box content

border	Container border.
padding	Inner padding.

Example – to set up a bottom container to have a one pixel dotted border and no padding:

```
.s7videoviewer .s7emaildialog .s7dialogbody .s7dialogcontent {
border: 1px dotted #A0A0A0;
padding: 0;
}
```

Thumbnail image is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogthumbnail
```

The background-image property is set by the component logic.

CSS properties of the dialog box thumbnail image

width	Thumbnail width.
height	Thumbnail height.
vertical-align	Vertical alignment thumbnail.
padding	Inner padding.

Example – to set up thumbnail to be 90 x 60 pixels, and top-aligned with ten pixels of padding:

```
.s7videoviewer .s7emaildialog .s7dialogthumbnail {
height: 60px;
padding: 10px;
vertical-align: top;
width: 90px;
}
```

Content title, origin, and description are further grouped into a panel to the right of the content thumbnail. It is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialoginfopanel
```

CSS properties of the dialog box information panel

width	Panel width.
-------	--------------

Example – to set up a content information panel to be 300 pixels wide:

```
.s7videoviewer .s7emaildialog .s7dialoginfopanel {
    width: 300px;
}
```

Content title is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogtitle
```

CSS properties of the dialog box title

margin	Outer margin.
font-weight	Font weight.
font-size	Font size.
font-family	Font family.

Example – to set up a content title to use bold font and have a ten pixel margin:

```
.s7videoviewer .s7emaildialog .s7dialogtitle {
    font-weight: bold;
    margin: 10px;
}
```

Content origin is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogorigin
```

CSS properties of the dialog box content origin

margin	Outer margin.
font-weight	Font weight.
font-size	Font size.
font-family	Font family.

Example – to set up content origin to have a ten pixel margin:

```
.s7videoviewer .s7emaildialog .s7dialogorigin {
    margin: 10px;
}
```

Content description is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogdescription
```

CSS properties of the dialog box content description

margin	Outer margin.
font-weight	Font weight.
font-size	Font size.
font-family	Font family.

Example – to set up a content description to have a ten pixel margin and use a nine point font:

```
.s7videoviewer .s7emaildialog .s7dialogdescription {
    font-size: 9pt;
    margin: 10px;
}
```

When a user enters incorrect input data and inline validation fails, or when the dialog box needs to render an error or a confirmation message when the form is submitted, a message is displayed in the top of the dialog box body. It is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogerrormessage
```

CSS properties of the dialog box error message

background-image	Error icon. Default is an exclamation mark.
background-position	Error icon position inside the message area.
color	Message text color.
font-weight	Font weight.
font-size	Font size.
font-family	Font family.
line-height	Text height inside the message. Affects vertical alignment.
padding	Inner padding.



Note: This message supports the *state* attribute selector with the following possible values: *verifyerror*, *senderror*, and *sendsuccess*. *verifyerror* is set when a message is displayed due to an inline input validation failure; *senderror* is set when a backend email service reports an error; *sendsuccess* is set when email is sent successfully. This way it is possible to style the message differently depending on the dialog box state.

The error message can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a message to use a ten point bold font, have 25 pixels line height, 20 pixels padding on the left, use an exclamation mark icon, red text in case of an error, and no icon and green text in case of success:

```
.s7videoviewer .s7emaildialog .s7dialogerrormessage[state="verifyerror"] {
    background-image: url("images/sdk/dlgerrimg.png");
    color: #FF0000;
}
.s7videoviewer .s7emaildialog .s7dialogerrormessage[state="senderror"] {
    background-image: url("images/sdk/dlgerrimg.png");
    color: #FF0000;
}
.s7videoviewer .s7emaildialog .s7dialogerrormessage[state="sendsuccess"] {
    background-image: none;
    color: #00B200;
}
.s7videoviewer .s7emaildialog .s7dialogerrormessage {
    background-position: left center;
    font-size: 10pt;
    font-weight: bold;
    line-height: 25px;
    padding-left: 20px;
}
```

If vertical scrolling is needed, the scroll bar is rendered in the panel near the right edge of the dialog, which is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7dialogscrollpanel
```

CSS properties of the dialog box scroll panel

width	Scroll panel width.
-------	---------------------

Example – to set up a scroll panel to be 44 pixels wide:

```
.s7videoviewer .s7emaildialog .s7dialogscrollpanel {
    width: 44px;
}
```

The appearance of the scroll bar area is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7scrollbar
```

CSS properties of the scroll bar

width	The scroll bar width.
top	The vertical scroll bar offset from the top of the scroll panel.
bottom	The vertical scroll bar offset from the bottom of the scroll panel.
right	The horizontal scroll bar offset from the right edge of the scroll panel.

Example – to set up a scroll bar that is 28 pixels wide, an eight pixel margin from the top, right, and bottom of the scroll panel:

```
.s7videoviewer .s7emaildialog .s7scrollbar {
    bottom: 8px;
    right: 8px;
    top: 8px;
    width: 28px;
}
```

Scroll bar track is the area between the top and bottom scroll buttons. The component automatically sets the position and height of the track. The track is controlled with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrolltrack
```

CSS properties of the scroll track

width	The track width.
background-color	The track background color.

Example – to set up a scroll bar track that is 28 pixels wide and has a gray background:

```
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrolltrack {
width:28px;
background-color: #B2B2B2;
}
```

Scroll bar thumb moves vertically within a scroll track area. Its vertical position is fully controlled by the component logic, however the thumb height does not dynamically change depending on the amount of content. You can configure the thumb height and other aspects with the following CSS class selector:

```
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollthumb
```

CSS properties of the scroll bar thumb

width	The thumb width.
height	The thumb height.
padding-top	The vertical padding between the top of the track.
padding-bottom	The vertical padding between the bottom of the track.
background-image	The image that is displayed for a given thumb state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: Thumb supports the *state* attribute selector, which can be used to apply different skins to different thumb states: up, down, over, and disabled.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up scroll bar thumb that is 28 x 45 pixels, has a ten pixel margin on the top and the bottom, and has different artwork for each state:

```
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollthumb {
height: 45px;
padding-bottom: 10px;
padding-top: 10px;
width: 28px;
}
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollthumb[state="up"] {
```


```
background-image:url("images/sdk/scrollbar_thumb_up.png");
}
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollthumb[state="down"] {
background-image:url("images/sdk/scrollbar_thumb_down.png");
}
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollthumb[state="over"] {
background-image:url("images/sdk/scrollbar_thumb_over.png");
}
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollthumb[state="disabled"] {
background-image:url("images/sdk/scrollbar_thumb_disabled.png");
}
```

The appearance of the top and bottom scroll buttons is controlled with the following CSS class selectors:

```
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollupbutton
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton
```

CSS properties of the top and bottom scroll buttons

width	The button width.
height	The button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

 **Note:** These buttons support the *state* attribute selector, which can be used to apply different skins to different button states: *up*, *down*, *over*, and *disabled*.

Example – to set up scroll buttons that are 28 x 32 pixels and have different artwork for each state:

```
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollupbutton {
width:28px;
height:32px;
}
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollupbutton[state='up'] {
background-image:url(images/sdk/scroll_up_up.png);
}
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollupbutton[state='over'] {
background-image:url(images/sdk/scroll_up_over.png);
}
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollupbutton[state='down'] {
background-image:url(images/sdk/scroll_up_down.png);
}
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollupbutton[state='disabled'] {
background-image:url(images/sdk/scroll_up_disabled.png);
}
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton {
width:28px;
height:32px;
}
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton[state='up'] {
background-image:url(images/sdk/scroll_down_up.png);
}
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrolldownbutton[state='over'] {
background-image:url(images/sdk/scroll_down_over.png);
}
```

```
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollbutton[state='down'] {
  background-image:url(images/sdk/scroll_down_down.png);
}
.s7videoviewer .s7emaildialog .s7scrollbar .s7scrollbutton[state='disabled'] {
  background-image:url(images/sdk/scroll_down_disabled.png);
}
```

Embed share

Embed share tool consists of a button added to the Social share panel and the modal dialog box that displays when the tool is activated. The position of the button is fully managed by the Social share tool.

The appearance of the embed share button is controlled with the following CSS class selector:

```
.s7videoviewer .s7embedshare
```

CSS properties of the embed share tool

width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a embed share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```
.s7videoviewer .s7embedshare {
  width:28px;
  height:28px;
}
.s7videoviewer .s7embedshare[state='up'] {
  background-image:url(images/v2/EmbedShare_dark_up.png);
}
.s7videoviewer .s7embedshare[state='over'] {
  background-image:url(images/v2/EmbedShare_dark_over.png);
}
.s7videoviewer .s7embedshare[state='down'] {
  background-image:url(images/v2/EmbedShare_dark_down.png);
}
.s7videoviewer .s7embedshare[state='disabled'] {
  background-image:url(images/v2/EmbedShare_dark_disabled.png);
}
```

The background overlay that covers the web page when the dialog box is active is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7backoverlay
```

CSS properties of the background overlay

opacity	Background overlay opacity.
background-color	Background overlay color.

Example – to set up a background overlay to be gray with 70% opacity:

```
.s7videoviewer .s7embeddialog .s7backoverlay {
  opacity:0.7;
  background-color:#222222;
}
```

By default the modal dialog box is displayed centered in the screen on desktop systems and takes the entire web page area on touch devices. In all cases, the positioning and sizing of the dialog box is managed by the component. The dialog box is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialog
```

CSS properties of the dialog box

border-radius	Dialog box border radius, in case the dialog box does not take the entire browser.
background-color	Dialog box background color.
width	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).
height	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).

Example – to set up the dialog box to use the entire browser window and have a white background on touch devices:

```
.s7videoviewer .s7touchinput .s7embeddialog .s7dialog {
  width:100%;
  height:100%;
  background-color: #ffffff;
}
```

Dialog box header consists of an icon, a title text, and a close button. The header container is controlled with

```
.s7videoviewer .s7embeddialog .s7dialogheader
```

CSS properties of the dialog box header

padding	Inner padding for header content.
---------	-----------------------------------

The icon and the title text are wrapped into an additional container controlled with

```
.s7videoviewer .s7embeddialog .s7dialogheader .s7dialogline
```

CSS properties of the dialog line

padding	Inner padding for the header icon and title
---------	---

Header icon is controlled with the following CSS class selector

```
.s7videoviewer .s7embeddialog .s7dialogheadericon
```

CSS properties of the dialog box header icon

width	Icon width.
height	Icon height.
background-image	Icon image.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

Header title is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialogheadertext
```

CSS properties of the dialog box header text

font-weight	Font weight.
font-size	Font height.
font-family	Font family.
padding	Internal text padding.


Close button is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7closebutton
```

CSS properties of the close button

top	Vertical button position relative to header container.
right	Horizontal button position relative to header container.
width	Button width.
height	Button height.
padding	Inner padding of button.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used.

	See CSS Sprites .
--	-----------------------------------

 **Note:** This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The Close button tool tip and the dialog box title can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up dialog header with padding, 24 x 14 pixels icon, bold 16 point title, and 28 x 28 pixels close button, positioned two pixels from the top, and two pixels from the right of dialog container:

```
.s7videoviewer .s7embeddialog .s7dialogheader {
padding: 10px;
}
.s7videoviewer .s7embeddialog .s7dialogheader .s7dialogline {
padding: 10px 10px 2px;
}
.s7videoviewer .s7embeddialog .s7dialogheadericon {
background-image: url("images/sdk/dlgembed_cap.png");
height: 14px;
width: 24px;
}
.s7videoviewer .s7embeddialog .s7dialogheadertext {
font-size: 16pt;
font-weight: bold;
padding-left: 16px;
}
.s7videoviewer .s7embeddialog .s7closebutton {
top: 2px;
right: 2px;
padding: 8px;
width: 28px;
height: 28px;
}
.s7videoviewer .s7embeddialog .s7closebutton[state='up'] {
background-image: url(images/sdk/close_up.png);
}
.s7videoviewer .s7embeddialog .s7closebutton[state='over'] {
background-image: url(images/sdk/close_over.png);
}
.s7videoviewer .s7embeddialog .s7closebutton[state='down'] {
background-image: url(images/sdk/close_down.png);
}
.s7videoviewer .s7embeddialog .s7closebutton[state='disabled'] {
background-image: url(images/sdk/close_disabled.png);
}
```

Dialog footer consists of "cancel" button. The footer container is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialogfooter
```

CSS properties of the dialog box footer

border	Border that you can use to visually separate the footer from the rest of the dialog box.
--------	--

The footer has an inner container which keeps the button. It is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialogbuttoncontainer
```

CSS properties of the dialog box button container

padding	Inner padding between the footer and the button.
---------	--

Select All button is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialogactionbutton
```

The button is only available on desktop systems.

CSS properties of the Select All button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: The Select All button supports the `state` attribute selector, which can be used to apply different skins to different button states.

Cancel button is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialogcancelbutton
```

CSS properties of the dialog box cancel button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: The cancel button supports the `state` attribute selector, which can be used to apply different skins to different button states.

In addition, both buttons share the same common CSS class which can contain CSS settings that are the same for other dialog box buttons:

```
.s7videoviewer .s7embeddialog .s7dialogfooter .s7button
```

CSS properties of the button

font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.

line-height	Text height inside the button. Affects vertical alignment.
box-shadow	Drop shadow.
margin-right	Right button margin.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box footer with a 64 x 34 Cancel button, having text color and background color different for each button state:

```
.s7videoviewer .s7embeddialog .s7dialogfooter {
  border-top: 1px solid #909090;
}
.s7videoviewer .s7embeddialog .s7dialogbuttoncontainer {
  padding-bottom: 6px;
  padding-top: 10px;
}
.s7videoviewer .s7embeddialog .s7dialogfooter .s7button {
  box-shadow: 1px 1px 1px #999999;
  color: #FFFFFF;
  font-size: 9pt;
  font-weight: bold;
  line-height: 34px;
  margin-right: 10px;
}
.s7videoviewer .s7embeddialog .s7dialogcancelbutton {
  width: 64px;
  height: 34px;
}
.s7videoviewer .s7embeddialog .s7dialogcancelbutton[state='up'] {
  background-color: #666666;
  color: #dddddd;
}
.s7videoviewer .s7embeddialog .s7dialogcancelbutton[state='down'] {
  background-color: #555555;
  color: #ffffff;
}
.s7videoviewer .s7embeddialog .s7dialogcancelbutton[state='over'] {
  background-color: #555555;
  color: #ffffff;
}
.s7videoviewer .s7embeddialog .s7dialogcancelbutton[state='disabled'] {
  background-color: #b2b2b2;
  color: #dddddd;
}
.s7videoviewer .s7embeddialog .s7dialogactionbutton {
  width: 82px;
  height: 34px;
}
.s7videoviewer .s7embeddialog .s7dialogactionbutton[state='up'] {
  background-color: #333333;
  color: #dddddd;
}
.s7videoviewer .s7embeddialog .s7dialogactionbutton[state='down'] {
  background-color: #222222;
  color: #cccccc;
}
.s7videoviewer .s7embeddialog .s7dialogactionbutton[state='over'] {
  background-color: #222222;
  color: #cccccc;
}
.s7videoviewer .s7embeddialog .s7dialogactionbutton[state='disabled'] {
  background-color: #b2b2b2;
}
```

```
color:#dddddd;
}
```

The main dialog area (between the header and the footer) contains scrollable dialog content and scroll panel on the right. In all cases, the component manages the width of this area, it is not possible to set it in CSS. Main dialog area is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialogviewarea
```

CSS properties of the dialog box viewing area

height	The height of the main dialog box area. It should be specified only when the dialog box works in desktop mode. It is not applicable when the dialog box is sized to occupy the entire browser window.
background-color	The background color of the main dialog box area.
margin	Outer margin.

Example – to set up a main dialog box area to be 300 pixels height, have a ten pixel margin, and use a white background:

```
.s7videoviewer .s7embeddialog .s7dialogviewarea {
background-color:#ffffff;
margin:10px;
height:300px;
}
```

All form content (like labels and input fields) resides inside a container controlled with

```
.s7videoviewer .s7embeddialog .s7dialogbody
```

If the height of this container appears to be bigger than the main dialog box area, a vertical scroll is enabled automatically by the component.

CSS properties of the dialog box body

padding	Inner padding.
---------	----------------

Example – to set up form content to have ten pixel padding:

```
.s7videoviewer .s7embeddialog .s7dialogbody {
padding: 10px;
}
```

All static labels in the dialog box form are controlled with

```
.s7videoviewer .s7embeddialog .s7dialoglabel
```

This class is not suitable for controlling the label size or position because you can apply it to texts in various places of the form user interface.

CSS properties of the dialog box label.

font-weight	Label font weight.
font-size	Label font size.

font-family	Label font family.
color	Label text color.

The dialog box labels tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up all labels to be gray, bold with a nine pixel font:

```
.s7videoviewer .s7embeddialog .s7dialoglabel {
  color: #666666;
  font-size: 9pt;
  font-weight: bold;
}
```

The size of the text copy displayed on top of the embed code is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialoginputwide
```

CSS properties of the dialog box input wide field

width	Input field width.
padding	Inner padding.

Example – to set text copy to be 430 pixels wide and have a ten pixel padding in the bottom:

```
.s7videoviewer .s7embeddialog .s7dialoginputwide {
  padding-bottom: 10px;
  width: 430px;
}
```

The embed code is wrapped into container and controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

width	The width of the embed code container.
border	Border around the embed code container.
padding	Inner padding.

Example – to set a one pixel grey border around embed code text, make it 430 pixels wide, and have a ten pixel padding:

```
.s7videoviewer .s7embeddialog .s7dialoginputcontainer {
  border: 1px solid #CCCCCC;
  padding: 10px;
  width: 430px;
}
```

The actual embed code text is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

word-wrap	Word wrapping style.
-----------	----------------------

Example – to set up embed code to use break-word word wrapping:

```
.s7videoviewer .s7embeddialog .s7dialogmessage {
  word-wrap: break-word;
}
```

Embed size label and drop-down are located in the bottom of the dialog box and put into a container controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialogembedsizepanel
```

CSS properties of the dialog box embed size panel

padding	Inner padding.
---------	----------------

Example – to set up an embed size panel to have ten pixels of padding:

```
.s7videoviewer .s7embeddialog .s7dialogembedsizepanel {
  padding: 10px;
}
```

The size and alignment of the embed size label is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialogembedsizelabel
```

CSS properties of the dialog box embed size panel

vertical-align	Vertical label alignment.
width	Label width.

Example – to set the embed size label to be top-aligned and 80 pixels wide:

```
.s7videoviewer .s7embeddialog .s7dialogembedsizelabel {
  vertical-align: top;
  width: 80px;
}
```

The width of the embed size combo box is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7combobox
```

CSS properties of the combo box

width	Combo box width.
-------	------------------



Note: The combo box supports the `expanded` attribute selector with possible values of `true` and `false`. `true` is used when combo box displays one of pre-defined embed sizes, thus should take all available width. `false` is used when custom size option is selected in the combo box, so it should shrink to allow space for custom width and height input fields.

Example – to set the embed size combo box to be 300 pixels wide when showing a pre-defined item and 110 pixels wide when showing a custom size:

```
.s7videoviewer .s7embeddialog .s7combobox[expanded="true"] {
  width: 300px;
}
```

```

}
.s7videoviewer .s7embeddialog .s7combobox[expanded="false"] {
    width: 110px;
}

```

The height of the combo box text is defined by a special inner element and is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7combobox .s7comboboxtext
```

CSS properties of the combo box text

height	Combo box text height.
--------	------------------------

Example – to set embed size combo box text height to 40 pixels:

```

.s7videoviewer .s7embeddialog .s7combobox .s7comboboxtext {
    height: 40px;
}

```

The combo box has a "drop down" button on the right and it is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7combobox .s7comboboxbutton
```

CSS properties of the combo box button

top	Vertical button position inside the combo box.
right	Horizontal button position inside the combo box.
width	Button width.
height	Button height.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

This button supports the state attribute selector, which can be used to apply different skins to different button states.

Example – to set a "drop down" button to 28 x 28 pixels and have a separate image for each state:

```

.s7videoviewer .s7embeddialog .s7combobox .s7comboboxbutton {
    width:28px;
    height:28px;
}
.s7videoviewer .s7embeddialog .s7combobox .s7comboboxbutton[state='up'] {
    background-image:url(images/sdk/cboxbtndn_up.png);
}
.s7videoviewer .s7embeddialog .s7combobox .s7comboboxbutton[state='over'] {
    background-image:url(images/sdk/cboxbtndn_over.png);
}
.s7videoviewer .s7embeddialog .s7combobox .s7comboboxbutton[state='down'] {
    background-image:url(images/sdk/cboxbtndn_over.png);
}
.s7videoviewer .s7embeddialog .s7combobox .s7comboboxbutton[state='disabled'] {
    background-image:url(images/sdk/cboxbtndn_up.png);
}

```


The panel with the list of embed sizes displayed when combo box is opened is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7comboboxdropdown
```

The size and position of the panel is controlled by the component. It is not possible to change it through CSS.

CSS properties of the combo box drop-down

border	Panel border.
--------	---------------

Example – to set the combo box panel to have a one pixel grey border:

```
.s7videoviewer .s7embeddialog .s7comboboxdropdown {
    border: 1px solid #CCCCCC;
}
```

A single item in a drop-down panel that is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dropdownitemanchor
```

CSS properties of the drop-down item anchor

background-color	Item background.
------------------	------------------

Example – to set the combo box panel item to have a white background:

```
.s7videoviewer .s7embeddialog .s7dropdownitemanchor {
    background-color: #FFFFFF;
}
```

A check mark displayed to the left of the selected item inside the combo box panel that is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7checkmark
```

CSS properties of the check mark box

width	Icon width.
height	Icon height.
background-image	Item image.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

Example – to set the check mark icon to 25 x 25 pixels:

```
.s7videoviewer .s7embeddialog .s7checkmark {
    background-image: url("images/sdk/cboxchecked.png");
    height: 25px;
    width: 25px;
}
```

When "Custom Size" option is selected in the embed size combo box the dialog box displays two extra input fields to the right to allow the user to enter a custom embed size. Those fields are wrapped in a container that is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialogcustomsizepanel
```

CSS properties of the dialog box custom size panel

left	Distance from the embed size combo box.
------	---

Example – to set custom size input fields panel to be 20 pixels to the right of the combo box:

```
.s7videoviewer .s7embeddialog .s7dialogcustomsizepanel {
    left: 20px;
}
```

Each custom size input field is wrapped in a container that renders a border and sets the margin between the fields. It is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialogcustomsize
```

CSS properties of the dialog box custom size

border	Border around the input field.
width	Input field width.
margin	Input field margin.
padding	Input field padding.

Example – to set the custom size input fields to have a one pixel grey border, margin, padding and be 70 pixels wide:

```
.s7videoviewer .s7embeddialog .s7dialogcustomsize {
    border: 1px solid #CCCCCC;
    margin-right: 20px;
    padding-left: 2px;
    padding-right: 2px;
    width: 70px;
}
```

If vertical scrolling is needed, the scroll bar is rendered in the panel near the right edge of the dialog box, which is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7dialogscrollpanel
```

CSS properties of the dialog box scroll panel

width	Scroll panel width.
-------	---------------------

Example – to set up a scroll panel to be 44 pixels wide

```
.s7videoviewer .s7embeddialog .s7dialogscrollpanel {
    width: 44px;
}
```

The appearance of scroll bar area is controlled with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7scrollbar
```

CSS properties of the scroll bar

width	Scroll bar width.
top	The vertical scroll bar offset from the top of the scroll panel.
bottom	The vertical scroll bar offset from the bottom of the scroll panel.
right	The horizontal scroll bar offset from the right edge of the scroll panel.

Example – to set up a scroll bar that is 28 pixels wide and has an eight pixel margin from the top, right, and bottom of the scroll panel:

```
.s7videoviewer .s7embeddialog .s7scrollbar {
  bottom: 8px;
  right: 8px;
  top: 8px;
  width: 28px;
}
```

Scroll bar track is the area between the top and bottom scroll buttons. The component automatically sets the position and height of the track. The track is controlled with the following CSS class selector

```
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrolltrack
```

CSS properties of the scroll bar track

width	Track width.
background-color	Track background color.

Example – to set up a scroll bar track that is 28 pixels wide and has a grey background:

```
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrolltrack {
width:28px;
background-color: #B2B2B2;
}
```

The scroll bar thumb moves vertically within a scroll track area. Its vertical position is fully controlled by the component logic. However, thumb height does not dynamically change depending on the amount of content. The thumb height and other aspects can be configured with the following CSS class selector:

```
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrollthumb
```

CSS properties of the scroll bar thumb

width	Thumb width.
height	Thumb height.
padding-top	The vertical padding between the top of the track.

padding-bottom	The vertical padding between the bottom of the track.
background-image	The image displayed for a given thumb state.



Note: Thumb supports the *state* attribute selector, which can be used to apply different skins to different thumb states: *up*, *down*, *over*, and *disabled*.

Example – to set up a scroll bar thumb that is 28 x 45 pixels, has a ten pixel margin on the top and bottom, and has different artwork for each state:

```
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrollthumb {
  height: 45px;
  padding-bottom: 10px;
  padding-top: 10px;
  width: 28px;
}
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrollthumb[state="up"] {
  background-image:url("images/sdk/scrollbar_thumb_up.png");
}
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrollthumb[state="down"] {
  background-image:url("images/sdk/scrollbar_thumb_down.png");
}
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrollthumb[state="over"] {
  background-image:url("images/sdk/scrollbar_thumb_over.png");
}
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrollthumb[state="disabled"] {
  background-image:url("images/sdk/scrollbar_thumb_disabled.png");
}
```

The appearance of the top and bottom scroll buttons is controlled with the following CSS class selectors:

```
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrollupbutton
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton
```

It is not possible to position scroll buttons using CSS top, left, bottom, and right properties. Instead, the viewer logic automatically positions them.

CSS properties of the top and bottom scroll buttons

width	Button width.
height	Button height.
background-image	The image displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: These buttons support the *state* attribute selector, which can be used to apply different skins to different button states: *up*, *down*, *over*, and *disabled*.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up scroll buttons that are 28 x 32 pixels and have different artwork for each state:

```
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrollupbutton {
  width:28px;
  height:32px;
}
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrollupbutton[state='up'] {
background-image:url(images/sdk/scroll_up_up.png);
}
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrollupbutton[state='over'] {
  background-image:url(images/sdk/scroll_up_over.png);
}
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrollupbutton[state='down'] {
  background-image:url(images/sdk/scroll_up_down.png);
}
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrollupbutton[state='disabled'] {
  background-image:url(images/sdk/scroll_up_disabled.png);
}
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton {
  width:28px;
  height:32px;
}
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton[state='up'] {
  background-image:url(images/sdk/scroll_down_up.png);
}
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton[state='over'] {
  background-image:url(images/sdk/scroll_down_over.png);
}
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton[state='down'] {
  background-image:url(images/sdk/scroll_down_down.png);
}
.s7videoviewer .s7embeddialog .s7scrollbar .s7scrolldownbutton[state='disabled'] {
  background-image:url(images/sdk/scroll_down_disabled.png);
}
```

Facebook share

Facebook share tool consists of a button added to the Social share panel. When the button is clicked the user is redirected to a sharing dialog box that is provided by a social service. The position of the button is fully managed by the Social share tool.

The appearance of the Facebook share button is controlled with the following CSS class selector:

```
.s7videoviewer .s7facebookshare
```

CSS properties of the Facebook share tool

width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Facebook share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```
.s7videoviewer .s7facebookshare {
  width:28px;
  height:28px;
}
.s7videoviewer .s7facebookshare[state='up'] {
background-image:url(images/v2/FacebookShare_dark_up.png);
}
.s7videoviewer .s7facebookshare[state='over'] {
background-image:url(images/v2/FacebookShare_dark_over.png);
}
.s7videoviewer .s7facebookshare[state='down'] {
background-image:url(images/v2/FacebookShare_dark_down.png);
}
.s7videoviewer .s7facebookshare[state='disabled'] {
background-image:url(images/v2/FacebookShare_dark_disabled.png);
}
```

Focus highlight

Input focus highlight displayed around focused viewer UI element is controlled with the CSS class selector.

CSS properties

The appearance is controlled with the following CSS class selector:

```
.s7videoviewer *:focus
```

CSS property	Description
outline	Focus highlight style.

Example – to disable the default browser focus highlight for all viewer user interface elements, add the following CSS selector to viewer's style sheet:

```
.s7videoviewer *:focus {
  outline: none;
}
```

Full screen button

The full screen button causes the video player to enter or exit full screen mode when a user clicks it.

You can size, skin, and position the full screen button, relative to the control bar that contains it, by CSS.

The appearance of the full screen button is controlled with the CSS class selector:

```
.s7videoviewer .s7fullscreenbutton
```

CSS properties of the full screen button

top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.

bottom	Position from the bottom border, including padding.
width	The width of the full screen button.
height	The height of the full screen button.
background-image	The displayed image for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the "full screen" state and *selected='false'* corresponds to the "normal" state.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example

To set up a full screen button that is 32 x 32 pixels, and positioned 6 pixels from the top and right edge of the control bar. Also, display a different image for each of the four different button states when selected or not selected.

```
.s7videoviewer .s7fullscreenbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7videoviewer .s7fullscreenbutton [selected='false'][state='up'] {
background-image:url(images/enterFullBtn_up.png);
}
.s7videoviewer .s7fullscreenbutton [selected='false'][state='over'] {
background-image:url(images/enterFullBtn_over.png);
}
.s7videoviewer .s7fullscreenbutton [selected='false'][state='down'] {
background-image:url(images/enterFullBtn_down.png);
}
.s7videoviewer .s7fullscreenbutton [selected='false'][state='disabled'] {
background-image:url(images/enterFullBtn_disabled.png);
}
.s7videoviewer .s7fullscreenbutton [selected='true'][state='up'] {
background-image:url(images/exitFullBtn_up.png);
}
.s7videoviewer .s7fullscreenbutton [selected='true'][state='over'] {
background-image:url(images/exitFullBtn_over.png);
}
.s7videoviewer .s7fullscreenbutton [selected='true'][state='down'] {
background-image:url(images/exitFullBtn_down.png);
}
.s7videoviewer .s7fullscreenbutton [selected='true'][state='disabled'] {
background-image:url(images/exitFullBtn_disabled.png);
}
```

Icon effect

The play icon is overlaid on the main view area. It displays when the video is paused, or when the end of the video is reached, and it also depends on the `iconeffect` parameter.

The appearance of the play icon is controlled with the following CSS class selector:

```
.s7videoviewer .s7videoplayer .s7iconeffect
```

CSS properties of the play icon

background-image	The displayed image for the play icon.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
width	The width of the play icon.
height	The height of the play icon.

Icon effect supports the `state` attribute selector. `state="play"` is used when the video is paused in the middle of playback, and `state="replay"` is used when the play head is in the end of the stream.

Example

Setup a 100 x 100 pixel play icon.

```
.s7videoviewer .s7videoplayer .s7iconeffect {
  width: 100px;
  height: 100px;}
.s7videoviewer .s7videoplayer .s7iconeffect[state="play"] {
  background-image: url(images/playIcon.png);
}
.s7videoviewer .s7videoplayer .s7iconeffect[state="replay"] {
  background-image: url(images/replayIcon.png);
}
```

Link share

Link share tool consists of a button added to the Social share panel and the modal dialog box that displays when the tool is activated. The position of the button is fully managed by the Social share tool.

The appearance of the link share button is controlled with the following CSS class selector:

```
.s7videoviewer .s7linkshare
```

CSS properties of the link share tool

width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.

background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
---------------------	---



Note: This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a link share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```
.s7videoviewer .s7linkshare {
  width:28px;
  height:28px;
}
.s7videoviewer .s7linkshare[state='up'] {
  background-image:url(images/v2/LinkShare_dark_up.png);
}
.s7videoviewer .s7linkshare[state='over'] {
  background-image:url(images/v2/LinkShare_dark_over.png);
}
.s7videoviewer .s7linkshare[state='down'] {
  background-image:url(images/v2/LinkShare_dark_down.png);
}
.s7videoviewer .s7linkshare[state='disabled'] {
  background-image:url(images/v2/LinkShare_dark_disabled.png);
}
```

The background overlay that covers the web page when the dialog box is active is controlled with the following CSS class selector:

```
.s7videoviewer .s7linkdialog .s7backoverlay
```

CSS properties of the background overlay

opacity	Background overlay opacity.
background-color	Background overlay color.

Example – to set up a background overlay to be gray with 70% opacity:

```
.s7videoviewer .s7linkdialog .s7backoverlay {
  opacity:0.7;
  background-color:#222222;
}
```

By default the modal dialog box is displayed centered in the screen on desktop systems and takes the entire web page area on touch devices. In all cases, the positioning and sizing of the dialog box is managed by the component. The dialog box is controlled with the following CSS class selector:

```
.s7videoviewer .s7linkdialog .s7dialog
```

CSS properties of the dialog box

border-radius	Dialog box border radius, in case the dialog box does not take the entire browser.
background-color	Dialog box background color.

width	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).
height	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).

Example – to set up the dialog box to use the entire browser window and have a white background on touch devices:

```
.s7videoviewer .s7touchinput .s7linkdialog .s7dialog {
  width:100%;
  height:100%;
  background-color: #ffffff;
}
```

Dialog box header consists of an icon, a title text, and a close button. The header container is controlled with

```
.s7videoviewer .s7linkdialog .s7dialogheader
```

CSS properties of the dialog box header

padding	Inner padding for header content.
---------	-----------------------------------

The icon and the title text are wrapped into an additional container controlled with

```
.s7videoviewer .s7linkdialog .s7dialogheader .s7dialogline
```

CSS properties of the dialog line

padding	Inner padding for the header icon and title
---------	---

Header icon is controlled with the following CSS class selector

```
.s7videoviewer .s7linkdialog .s7dialogheadericon
```

CSS properties of the dialog box header icon

width	Icon width.
height	Icon height.
background-image	Icon image.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

Header title is controlled with the following CSS class selector:

```
.s7videoviewer .s7linkdialog .s7dialogheadertext
```

CSS properties of the dialog box header text

font-weight	Font weight.
-------------	--------------

font-size	Font height.
font-family	Font family.
padding	Internal text padding.

Close button is controlled with the following CSS class selector:

```
.s7videoviewer .s7linkdialog .s7closebutton
```

CSS properties of the close button

top	Vertical button position relative to header container.
right	Horizontal button position relative to header container.
width	Button width.
height	Button height.
padding	Inner padding of button.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The Close button tool tip and the dialog box title can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box header with padding, 22 x 12 pixels icon, bold 16 point title, and a 28 x 28 pixel Close button that is positioned two pixels from the top and two pixels from the right of the dialog box container:

```
.s7videoviewer .s7linkdialog .s7dialogheader {
  padding: 10px;
}
.s7videoviewer .s7linkdialog .s7dialogheader .s7dialogline {
  padding: 10px 10px 2px;
}
.s7videoviewer .s7linkdialog .s7dialogheadericon {
  background-image: url("images/sdk/dlglink_cap.png");
  height: 12px;
  width: 22px;
}
.s7videoviewer .s7linkdialog .s7dialogheadertext {
  font-size: 16pt;
  font-weight: bold;
  padding-left: 16px;
}
.s7videoviewer .s7linkdialog .s7closebutton {
  top: 2px;
  right: 2px;
```

```
padding:8px;
width:28px;
height:28px;
}
.s7videoviewer .s7linkdialog .s7closebutton[state='up'] {
background-image:url(images/sdk/close_up.png);
}
.s7videoviewer .s7linkdialog .s7closebutton[state='over'] {
background-image:url(images/sdk/close_over.png);
}
.s7videoviewer .s7linkdialog .s7closebutton[state='down'] {
background-image:url(images/sdk/close_down.png);
}
.s7videoviewer .s7linkdialog .s7closebutton[state='disabled'] {
background-image:url(images/sdk/close_disabled.png);
}
```

Dialog footer consists of "cancel" button. The footer container is controlled with the following CSS class selector:

```
.s7videoviewer .s7linkdialog .s7dialogfooter
```

CSS properties of the dialog box footer

border	Border that you can use to visually separate the footer from the rest of the dialog box.
--------	--

The footer has an inner container which keeps the button. It is controlled with the following CSS class selector:

```
.s7videoviewer .s7linkdialog .s7dialogbuttoncontainer
```

CSS properties of the dialog box button container

padding	Inner padding between the footer and the button.
---------	--


Select All button is controlled with the following CSS class selector:

```
.s7videoviewer .s7linkdialog .s7dialogactionbutton
```

The button is only available on desktop systems.

CSS properties of the Select All button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.

 **Note:** The Select All button supports the *state* attribute selector, which can be used to apply different skins to different button states.

Cancel button is controlled with the following CSS class selector:

```
.s7videoviewer .s7linkdialog .s7dialogcancelbutton
```

CSS properties of the dialog box cancel button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

In addition, both buttons share the same common CSS class which can contain CSS settings that are the same for other dialog box buttons:

```
.s7videoviewer .s7linkdialog .s7dialogfooter .s7button
```

CSS properties of the button

font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.
line-height	Text height inside the button. Affects vertical alignment.
box-shadow	Drop shadow.
margin-right	Right button margin.

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a dialog box footer with a 64 x 34 Cancel button, having text color and background color different for each button state:

```
.s7videoviewer .s7linkdialog .s7dialogfooter {
  border-top: 1px solid #909090;
}
.s7videoviewer .s7linkdialog .s7dialogbuttoncontainer {
  padding-bottom: 6px;
  padding-top: 10px;
}
.s7videoviewer .s7linkdialog .s7dialogfooter .s7button {
  box-shadow: 1px 1px 1px #999999;
  color: #FFFFFF;
  font-size: 9pt;
  font-weight: bold;
  line-height: 34px;
  margin-right: 10px;
}
.s7videoviewer .s7linkdialog .s7dialogcancelbutton {
  width: 64px;
  height: 34px;
}
```

```
.s7videoviewer .s7linkdialog .s7dialogcancelbutton[state='up'] {
  background-color:#666666;
  color:#dddddd;
}
.s7videoviewer .s7linkdialog .s7dialogcancelbutton[state='down'] {
  background-color:#555555;
  color:#ffffff;
}
.s7videoviewer .s7linkdialog .s7dialogcancelbutton[state='over'] {
  background-color:#555555;
  color:#ffffff;
}
.s7videoviewer .s7linkdialog .s7dialogcancelbutton[state='disabled'] {
  background-color:#b2b2b2;
  color:#dddddd;
}
.s7videoviewer .s7linkdialog .s7dialogactionbutton {
  width:82px;
  height:34px;
}
.s7videoviewer .s7linkdialog .s7dialogactionbutton[state='up'] {
  background-color:#333333;
  color:#dddddd;
}
.s7videoviewer .s7linkdialog .s7dialogactionbutton[state='down'] {
  background-color:#222222;
  color:#cccccc;
}
.s7videoviewer .s7linkdialog .s7dialogactionbutton[state='over'] {
  background-color:#222222;
  color:#cccccc;
}
.s7videoviewer .s7linkdialog .s7dialogactionbutton[state='disabled'] {
  background-color:#b2b2b2;
  color:#dddddd;
}
```

The main dialog area (between the header and the footer) contains dialog content. In all cases, the component manages the width of this area—it is not possible to set it in CSS. Main dialog area is controlled with the following CSS class selector:

```
.s7videoviewer .s7linkdialog .s7dialogviewarea
```

CSS properties of the dialog box viewing area

height	The height of the main dialog box area. It should be specified only when the dialog box works in desktop mode. It is not applicable when the dialog box is sized to occupy the entire browser window.
background-color	The background color of the main dialog box area.
margin	Outer margin.

Example – to set up a main dialog box area to be 300 pixels height, have a ten pixel margin, and use a white background:

```
.s7videoviewer .s7linkdialog .s7dialogviewarea {
  background-color:#ffffff;
  margin:10px;
  height:300px;
}
```

All form content (like labels and input fields) resides inside a container controlled with

```
.s7videoviewer .s7linkdialog .s7dialogbody
```

CSS properties of the dialog box body

padding	Inner padding.
---------	----------------

Example – to set up form content to have ten pixel padding:

```
.s7videoviewer .s7linkdialog .s7dialogbody {
  padding: 10px;
}
```

All static labels in the dialog box form are controlled with

```
.s7videoviewer .s7linkdialog .s7dialoglabel
```

This class is not suitable for controlling the label size or position because you can apply it to texts in various places of the form user interface.

CSS properties of the dialog box label.

font-weight	Label font weight.
font-size	Label font size.
font-family	Label font family.
color	Label text color.

The dialog box labels can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up all labels to be gray, bold with a nine pixel font:

```
.s7videoviewer .s7linkdialog .s7dialoglabel {
  color: #666666;
  font-size: 9pt;
  font-weight: bold;
}
```

The size of the text copy displayed on top of the link is controlled with the following CSS class selector:

```
.s7videoviewer .s7linkdialog .s7dialoginputwide
```

CSS properties of the dialog box input wide field

width	Text width.
padding	Inner padding.

Example – to set text copy to be 430 pixels wide and have a ten pixel padding in the bottom:

```
.s7videoviewer .s7linkdialog .s7dialoginputwide {
  padding-bottom: 10px;
  width: 430px;
}
```

The share link is wrapped in a container and controlled with the following CSS class selector:

```
.s7videoviewer .s7linkdialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

border	Border around the share link container.
padding	Inner padding.

Example – to set a one pixel grey border around embed code text and have nine pixels of padding:

```
.s7videoviewer .s7linkdialog .s7dialoginputcontainer {
  border: 1px solid #CCCCCC;
  padding: 9px;
}
```

The share link itself is controlled with the following CSS class selector:

```
.s7videoviewer .s7linkdialog .s7dialoglink
```

CSS properties of the dialog box share link

width	Share link width.
-------	-------------------

Example – to set the share link to be 450 pixels wide:

```
.s7videoviewer .s7linkdialog .s7dialoglink {
  width: 450px;
}
```

Main viewer area

The main view area is occupied by the video. It usually sets to fit the available device screen when no size is specified.

The following CSS class selector controls the appearance of the viewing area:

```
.s7videoviewer
```

CSS properties of the main viewer area

width	Viewer width.
height	Viewer height.
background-color	Background color in hexadecimal format.

Example

To set up a video viewer with a white background (#FFFFFF) and make its size 512 x 288 pixels:

```
.s7videoviewer {
  background-color: #FFFFFF;
  width: 512px;
  height: 288px;
}
```

Mutable volume

The mutable volume control initially appears as a button that lets a user mute or unmute the video player sound.

When a user rolls over the button, a slider appears that allows a user to set the volume. The mutable volume control can be sized, skinned, and positioned, relative to the control bar that contains it, by CSS.

The appearance of the mutable volume area is controlled with the following CSS class selector:

```
.s7videoviewer .s7mutablevolume
```

CSS properties of the mutable volume

top	Position from the top border, including padding.
right	Position from the right border, including padding.
width	The width of the mutable volume control.
height	The height of the mutable volume control.
background-color	The color of the mutable volume control.

The mute/unmute button appearance is controlled with the following CSS class selector:

```
.s7videoviewer .s7mutablevolume .s7mutebutton
```

You can control background image for each button state. The size of the button is inherited from the size of the volume control.

CSS properties of the button image

background-image	The image displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the "muted" state and *selected='false'* corresponds to the "unmuted" state.

The vertical volume bar area is controlled with the following CSS class selector:

```
.s7videoviewer .s7mutablevolume .s7verticalvolume
```

CSS properties of the vertical volume bar area

background-color	The background color of the vertical volume.
width	The width of the vertical volume.
height	The height of the vertical volume.

The track inside vertical volume control is controlled with the following CSS class selectors:

```
.s7videoviewer .s7mutablevolume .s7verticalvolume .s7track
.s7videoviewer .s7mutablevolume .s7verticalvolume .s7filledtrack
```

CSS properties of the vertical volume control

background-color	The background color of the vertical volume control.
width	Width of the vertical volume control.
height	Height of the vertical volume control.

The vertical volume knob is controlled with the following CSS class selector:

```
.s7videoviewer .s7mutablevolume .s7verticalvolume .s7knob
```

CSS properties of the vertical volume control knob

background-image	Vertical volume control knob artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
width	Width of the vertical volume control knob.
height	Height of the vertical volume control knob.
left	Horizontal position of the vertical volume control knob.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Examples

To set up a mute button that is 32 x 32 pixels and positioned 6 pixels from the top, and 38 pixels from the right edge of the control bar. Display a different image for each of the four different button states when selected or not selected.

```
.s7videoviewer .s7mutablevolume {
top:6px;
right:38px;
width:32px;
height:32px;
}
.s7videoviewer .s7mutablevolume .s7mutebutton[selected='true'][state='up'] {
background-image:url(images/mute_up.png);
}
.s7videoviewer .s7mutablevolume .s7mutebutton[selected='true'][state='over'] {
background-image:url(images/mute_over.png);
}
.s7videoviewer .s7mutablevolume .s7mutebutton[selected='true'][state='down'] {
background-image:url(images/mute_down.png);
}
.s7videoviewer .s7mutablevolume .s7mutebutton[selected='true'][state='disabled'] {
background-image:url(images/mute_disabled.png);
}
.s7videoviewer .s7mutablevolume .s7mutebutton[selected='false'][state='up'] {
```

```
background-image:url(images/unmute_up.png);
}
.s7videoviewer .s7mutablevolume .s7mutebutton[selected='false'][state='over'] {
background-image:url(images/unmute_over.png);
}
.s7videoviewer .s7mutablevolume .s7mutebutton[selected='false'][state='down'] {
background-image:url(images/unmute_down.png);
}
.s7videoviewer .s7mutablevolume .s7mutebutton[selected='false'][state='disabled'] {
background-image:url(images/unmute_disabled.png);
}
```

The following is an example of how you can style the volume slider within the mutable volume control.

```
.s7videoviewer .s7mutablevolume .s7verticalvolume {
width:36px;
height:83px;
left:0px;
background-color:#dddddd;
}
.s7videoviewer .s7mutablevolume .s7verticalvolume .s7track {
top:11px;
left:14px;
width:10px;
height:63px;
background-color:#666666;
}
.s7videoviewer .s7mutablevolume .s7verticalvolume .s7filledtrack {
width:10px;
background-color:#ababab;
}
.s7videoviewer .s7mutablevolume .s7verticalvolume .s7knob {
width:18px;
height:10px;
left:9px;
background-image:url(images/volumeKnob.png);
}
```

The following is an example of how you can customize the video player so sound is disabled during playback. Add the following code to the viewer's embed code:

```
    "handlers":{
        "initComplete":function() {
            videoViewer.getComponent("mutableVolume").setPosition(0);
            videoViewer.getComponent("mutableVolume").deactivate();
        }
    }
```

In the code example above, the volume level is set to 0 on the mutableVolume component. Then, the same component is deactivated so it cannot be used by the end user.

Play/Pause button

The play/pause button causes the video player to play or pause the video content when a user clicks it.

You can size, skin, and position the button, relative to the control bar that contains it, by CSS.

The following CSS class selector controls the appearance of the button:

```
.s7videoviewer .s7playpausebutton
```

CSS properties of the play/pause button

top	Position from the top border, including padding.
-----	--

right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports both the `state`, `selected`, and `replay` attribute selectors, which can be used to apply different skins to different button states. In particular, `selected='true'` corresponds to the "play" state and `selected='false'` corresponds to the "pause" state;

`replay='true'` is set when the video has reached the end and clicking on the button restarts playback from the beginning.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example

To set up a play/pause button that is 32 x 32 pixels; it is positioned six pixels from the top and left edge of the control bar, and displays a different image for each of the four different button states when selected or not selected.

```
.s7videoviewer .s7playpausebutton {
top:6px;
left:6px;
width:32px;
height:32px;
}
.s7videoviewer .s7playpausebutton[selected='true'][state='up'] {
background-image:url(images/playBtn_up.png);
}
.s7videoviewer .s7playpausebutton[selected='true'][state='over'] {
background-image:url(images/playBtn_over.png);
}
.s7videoviewer .s7playpausebutton[selected='true'][state='down'] {
background-image:url(images/playBtn_down.png);
}
.s7videoviewer .s7playpausebutton[selected='true'][state='disabled'] {
background-image:url(images/playBtn_disabled.png);
}
.s7videoviewer .s7playpausebutton[selected='false'][state='up'] {
background-image:url(images/pauseBtn_up.png);
}
.s7videoviewer .s7playpausebutton[selected='false'][state='over'] {
background-image:url(images/pauseBtn_over.png);
}
.s7videoviewer .s7playpausebutton[selected='false'][state='down'] {
background-image:url(images/pauseBtn_down.png);
}
```

```
.s7videoviewer .s7playpausebutton[selected='false'][state='disabled'] {
background-image:url(images/pauseBtn_disabled.png); }
}
.s7videoviewer .s7playpausebutton[selected='true'][replay='true'][state='up'] {
background-image:url(images/replayBtn_up.png); }
}
.s7videoviewer .s7playpausebutton[selected='true'][replay='true'][state='over'] {
background-image:url(images/replayBtn_over.png); }
}
.s7videoviewer .s7playpausebutton[selected='true'][replay='true'][state='down'] {
background-image:url(images/replayBtn_down.png); }
}
.s7videoviewer .s7playpausebutton[selected='true'][replay='true'][state='disabled'] {
background-image:url(images/replayBtn_disabled.png); }
}
```

Social share

The social share tool appears in the top right corner by default. It consists of a button and a panel which expands when user clicks or taps on a button and contains individual sharing tools.

The position and size of the social share tool in the viewer user interface is controlled with the following:

```
.s7videoviewer .s7socialshare
```

CSS properties of the social share tool

top	Vertical position of the social sharing tool relative to the viewer container.
left	Horizontal position of the social sharing tool relative to the viewer container.
width	The width of the social sharing tool.
height	The height of the social sharing tool.

Example – set up a social sharing tool that is positioned four pixels from the top and five pixels from the right of viewer container and is sized to 28 x 28 pixels.


```
.s7videoviewer .s7socialshare {
top:4px;
right:5px;
width:28px;
height:28px;
}
```

The appearance of the social share tool button is controlled with the following CSS class selector:

```
.s7videoviewer .s7socialshare .s7socialbutton
```

CSS properties of the social button

background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

 **Note:** This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – set up a social sharing tool button that displays a different image for each of the four different button states.

```
.s7videoviewer .s7socialshare .s7socialbutton[state='up'] {
background-image:url(images/v2/SocialShare_video_dark_up.png);
}
.s7videoviewer .s7socialshare .s7socialbutton[state='over'] {
background-image:url(images/v2/SocialShare_dark_over.png);
}
.s7videoviewer .s7socialshare .s7socialbutton[state='down'] {
background-image:url(images/v2/SocialShare_dark_down.png);
}
.s7videoviewer .s7socialshare .s7socialbutton[state='disabled'] {
background-image:url(images/v2/SocialShare_dark_disabled.png);
}
```

The appearance of the panel which contains the individual social sharing icons is controlled with the following CSS class selector:

```
.s7videoviewer .s7socialshare .s7socialsharepanel
```

CSS properties of the social share panel

background-color	The background color of the panel.
------------------	------------------------------------

Example – set up a panel to have transparent color:

```
.s7videoviewer .s7socialshare .s7socialsharepanel {
background-color: transparent;
}
```

Tooltips

On desktop systems some user interface elements like buttons have tooltips that are displayed on mouse hover.

CSS properties of the main viewer area

The appearance of tooltips is controlled with the following CSS class selector:

```
.s7tooltip
```

CSS property	Description
border-radius	Background border radius.
border-color	Background border color.
background-color	Background color.
color	Text color.
font-family	Text font name.
font-size	Text font size.



Note: In case tooltip styles are customized from within the embedding web page, all properties have to contain !IMPORTANT rule. This is not necessary if tooltips are customized in the viewer's CSS file.

Example – to set up tooltips that have a grey border with 3px corner radius, black background and white text written with Arial, 11 pixels size:

```
.s7tooltip {
  border-radius: 3px 3px 3px 3px;
  border-color: #999999;
  background-color: #000000;
  color: #FFFFFF;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 11px;
}
```

Twitter share

Twitter share tool consists of a button added to the Social share panel. When the button is clicked the user is redirected to a sharing dialog box that is provided by a social service. The position of the button is fully managed by the Social share tool.

The appearance of the Twitter share button is controlled with the following CSS class selector:

```
.s7videoviewer .s7twittershare
```

CSS properties of the Twitter share tool

width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a Twitter share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```
.s7videoviewer .s7twittershare {
  width:28px;
  height:28px;
}
.s7videoviewer .s7twittershare[state='up'] {
  background-image:url(images/v2/TwitterShare_dark_up.png);
}
.s7videoviewer .s7twittershare[state='over'] {
  background-image:url(images/v2/TwitterShare_dark_over.png);
}
.s7videoviewer .s7twittershare[state='down'] {
  background-image:url(images/v2/TwitterShare_dark_down.png);
}
```

```
.s7videoviewer .s7twittershare[state='disabled'] {
background-image:url(images/v2/TwitterShare_dark_disabled.png);
}
```

Video player

The video player is the rectangular area where the video content is displayed within the viewer.

If the dimensions of the video that is being played does not match the dimensions of the video player, the video content is centered within the video player's rectangle display area.

The following CSS class selector controls the appearance of the video player:

```
.s7videoviewer .s7videoplayer
```

CSS properties of the video player

background-color	Background color of the main view.
------------------	------------------------------------

The error message that is displayed if the system is not capable of playing the video can be localized. See [Localization of user interface elements](#) for more information.

Example – To set up a video viewer with the video player size set to 512 x 288 pixels.

```
.s7videoviewer .s7videoplayer{
background-color: transparent;
}
```

Closed captions are put into an internal container inside the video player. The position of that container is controlled by supported WebVTT positioning operators. The caption text itself is inside that container, and its style is controlled with the following CSS class selector:

```
. s7videoviewer .s7 videoplayer .s7caption
```

CSS properties of closed captioning

background-color	Closed caption text background.
color	Close caption text color.
font-weight	Closed caption font weight.
font-size	Closed caption font size.
font-family	Closed caption font.

Example – To set up closed caption text to be 14 pixels, light gray, Arial, on a semi-transparent black background:

```
.s7videoviewer .s7videoplayer .s7caption {
background-color: rgba(0,0,0,0.75);
color: #e6e6e6;
font-weight: normal;
font-size: 14px;
font-family: Arial,Helvetica,sans-serif;
}
```


The appearance of the buffering animation is controlled with the following CSS class selector:

```
.s7videoviewer .s7videoplayer .s7waiticon
```

CSS properties of wait icon

CSS property	Description
width	Animation icon width.
height	Animation icon height.
margin-left	Animation icon left margin, normally minus half of the icon's width.
margin-top	Animation icon top margin, normally minus half of the icon's height.
background-image	Knob artwork.

Example – to set up a buffering animation to be 101 pixels wide, 29 pixels high:

```
.s7videoviewer .s7videoplayer .s7waiticon {
  width: 101px;
  height: 29px;
  margin-left: -50px;
  margin-top: -15px;
  background-image: url(images/sdk/busyicon.gif);
}
```

Video scrubber

The video scrubber is the horizontal slider control that lets a user dynamically seek to any time position within the currently playing video.

The scrubber 'knob' also moves as the video plays to indicate the current time position of the video during playback. The video scrubber always takes the whole width of the control bar. It is possible to skin the video scrubber. change its height and vertical position, by CSS.

The general appearance of the video scrubber is controlled with the following CSS class selector:

```
.s7videoviewer .s7videoscrubber
.s7videoviewer .s7videoscrubber .s7videotime
.s7videoviewer .s7videoscrubber .s7knob
```

CSS properties of the video scrubber

top	Position from the top border, including padding.
bottom	Position from the bottom border, including padding.
height	Height of the video scrubber.

background-color	The color of the video scrubber.
------------------	----------------------------------

The following CSS class selectors track background, play, and load indicators:

```
.s7videoviewer .s7videoscrubber .s7track
.s7videoviewer .s7videoscrubber .s7trackloaded
.s7videoviewer .s7videoscrubber .s7trackplayed
```

CSS properties of the track

height	Height of the corresponding track.
background-color	The color of the corresponding track.

The following CSS class selector controls the knob:

```
.s7videoviewer .s7videoscrubber .s7knob
```

CSS properties of the knob

top	Vertical knob offset.
width	Width of knob.
height	Height of knob.
background-image	Knob artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

The following CSS class selector controls the time played bubble:

```
.s7videoviewer .s7videoscrubber .s7videotime
```

CSS properties of the time played bubble

font-family	The font family to use for the time display text.
font-size	The font size to use for the time display text.
color	The font color to use for the time display text.
width	Bubble area width.
height	Bubble area height.
padding	Bubble area padding.

background-image	Bubble artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
text-align	Alignment of text with the bubble area.

The video scrubber tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – To set up a video viewer with a video scrubber with custom track colors that is 10 pixels tall, and positioned 10 pixels and 35 pixels from the top and left edges of the control bar.

```
.s7videoviewer .s7videoscrubber {
top:10px;
left:35px;
height:10px;
background-color:#AAAAAA;
}
.s7videoviewer .s7videoscrubber .s7track {
height:10px;
background-color:#444444;
}
.s7videoviewer .s7videoscrubber .s7trackloaded {
height:10px;
background-color:#666666;
}
.s7videoviewer .s7videoscrubber .s7trackplayed {
height:10px;
background-color:#888888;
}
```

When video chaptering is enabled with the `navigation` parameter, chapter locations are displayed as markers on top of the video scrubber track.

The video chapter marker is controlled by the following CSS class selector:

```
.s7videoviewer .s7videoscrubber .s7navigation
```

CSS properties of the video chapter marker

width	Video chapter marker width.
height	Video chapter marker height.
background-image	Video chapter marker artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports both the `state` attribute selector, which you can use to apply different skins to different button states. In particular, `selected='default'` corresponds to the default video chapter marker state and `selected='over'` is used when the video chapter marker is activated by a mouse over or touch gesture.

Example – To set up a video chapter marker that is 5 x 8 pixels and uses different art for the "default" and "over" state.

```
.s7videoviewer .s7videoscubber .s7navigation {
width:5px;
height:8px;
}
.s7videoviewer .s7videoscubber .s7navigation[state="default"] {
background-image: url("images/v2/VideoScrubberDiamond.png");
}
.s7videoviewer .s7videoscubber .s7navigation[state="over"] {
background-image: url("images/v2/VideoScrubberDiamond_over.png");
}
```

Video chapter bubble is positioned on top of the video chapter marker and shows the title, start time, and description for a given chapter. It is possible to control the maximum bubble width and vertical offset relative to the video scrubber track. The rest is calculated automatically by the component.

The video chapter bubble is controlled by the following CSS class selector:

```
.s7videoviewer .s7videoscubber .s7chapter
```

CSS properties of the video chapter bubble

max-width	Maximum width of the video chapter bubble.
bottom	Vertical offset from the video scrubber track.

Example – To set up a video chapter bubble that is 235 pixels wide and is eight pixels up from the bottom of the video scrubber track.

```
.s7videoviewer .s7videoscubber .s7chapter {
max-width:235px;
bottom:8px;
}
```

The video chapter bubble consists of an optional header and content. The header has the optional chapter start time and chapter title.

The header is controlled by the following CSS class selector:

```
.s7videoviewer .s7videoscubber .s7chapter .s7header
```

CSS properties of the video chapter bubble header

height	Video chapter bubble header height.
padding	Inner padding for video chapter bubble header text.
background-color	Video chapter bubble header background color.
line-height	Video chapter bubble header text line height.

Example – To set up a video chapter bubble header that is 22 pixels high, a 22 pixel line height, a 12 pixel horizontal margin, and a gray background.

```
.s7videoviewer .s7videoscubber .s7chapter .s7header {
height:22px;
padding:0 12px;
```

```
line-height:22px;
background-color: rgba(51, 51, 51, 0.8);
}
```

The start time of the video chapter is controlled by the following CSS class selector:

```
.s7videoviewer .s7videoscubber .s7chapter .s7header .s7starttime
```

CSS properties of the video chapter start time

color	Text color.
font-weight	Font weight.
font-size	Font size.
font-family	Font family.
padding-right	Padding between the start time and chapter title.

Example – To set up chapter start time using gray ten pixels Verdana font and has ten pixels padding to the right.

```
.s7videoviewer .s7videoscubber .s7chapter .s7header .s7starttime {
color: #dddddd;
font-family: Verdana,Arial,Helvetica,sans-serif;
font-size: 10px;
padding-right: 10px;
}
```

The video chapter title is controlled by the following CSS class selector:

```
.s7videoviewer .s7videoscubber .s7chapter .s7header .s7title
```

CSS properties of the video chapter title

color	Video chapter title text color.
font-weight	Video chapter title font weight.
font-size	Video chapter title font size.
font-family	Video chapter title font family.

Example – To set up a video chapter title that uses a white, bold, ten pixel Verdana font.

```
.s7videoviewer .s7videoscubber .s7chapter .s7header .s7title {
color: #ffffff;
font-family: Verdana,Arial,Helvetica,sans-serif;
font-size: 10px;
font-weight: bold;
}
```

The video chapter description is controlled by the following CSS class selector:

```
.s7videoviewer .s7videoscubber .s7chapter .s7description
```

CSS properties of the video chapter description

color	Video chapter description text color.
background-color	Video chapter description background color.
font-weight	Video chapter description font weight.
font-size	Video chapter description font size.
font-family	Video chapter description font family.
line-height	Video chapter description line height.
padding	Video chapter description inner padding.

Example – To set up video chapter description using a dark gray, 11 pixel Verdana font, with a light gray background; 5 pixel line height, 12 pixel horizontal padding, 12 pixel top padding, and 9 pixel bottom padding.

```
.s7videoviewer .s7videoscubber .s7chapter .s7description {
color: #333333;
background-color: rgba(221, 221, 221, 0.9);
font-family: Verdana,Arial,Helvetica,sans-serif;
font-size: 11px;
line-height: 15px;
padding: 12px 12px 9px;
}
```

The wedge connector within the bottom of the chapter bubble is controlled by the following CSS class selector:

```
.s7videoviewer .s7videoscubber .s7chapter .s7tail
```

CSS properties of the wedge connector

border-color	Wedge connector color. Defined as <color> transparent transparent so that only the top border color is defined and the remaining borders are left transparent.
border-width	Wedge connector width. Defined as <width> <width> 0 so that the same width is defined for the top and horizontal borders only and the bottom border width is 0.
margin	Defines a negative bottom margin only. It should have the same value as that of border-width.

Example – To set up a gray, six pixel wedge connector:

```
.s7videoviewer .s7videoscubber .s7chapter .s7tail {
border-color: rgba(221, 221, 221, 0.9) transparent transparent;
border-width: 6px 6px 0;
margin: 0 0 0 -6px;
}
```

Video time

The video time is the numeric display that shows the current time and duration of the currently playing video.

The video time font family, font size, and font color are among the properties that CSS can control. It can also be positioned, relative to the control bar that contains it, by CSS.

The appearance of the video time is controlled with the following CSS class selector:

```
.s7videoviewer .s7videotime
```

CSS properties of video time

top	Position from the top border, including padding.
right	Position from the right border, including padding.
width	The width of video time control. This property is required for Internet Explorer 8 or greater to function properly.
font-family	The font family to use for the time display text.
font-size	The font size to use for the time display text.
color	The font color to use for the time display text.

Example

Set the video time to light gray (hexadecimal #BBBBBB), sized at 12 pixels, positioned 15 pixels from the top of the control bar, and 80 pixels from the right edges of the control bar.

```
.s7videoviewer .s7videotime {  
top:15px;  
right:80px;  
font-size:12px;  
color:#BBBBBB;  
width:60px;  
}
```

Support for Adobe Analytics tracking

The Video Viewer supports Adobe Analytics tracking out-of-the-box.

Out-of-the-box tracking

The Video Viewer supports Adobe Analytics tracking out-of-the-box.

To enable tracking, pass the proper company preset name as `config2` parameter.

The viewer also sends a single tracking HTTP request to the configured Image Server with the viewer type and version information.

Custom tracking

To integrate with third-party analytics systems it is necessary to listen to `trackEvent` viewer callback and process `eventInfo` argument of the callback function as necessary. The following code is an example of such handler function:

```
var videoViewer = new s7viewers.VideoViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/Glacier_Climber_MP4",
    "serverurl": "http://s7dl.scene7.com/is/image/",
    "videoserverurl": "http://s7dl.scene7.com/is/content/"
  },
  "handlers": {
    "trackEvent": function(objID, compClass, instName, timeStamp, eventInfo) {
      //identify event type
      var eventType = eventInfo.split(",")[0];
      switch (eventType) {
        case "LOAD":
          //custom event processing code
          break;
        //additional cases for other events
      }
    }
  }
});
```

The viewer tracks the following SDK user events:

SDK user event	Sent when...
LOAD	viewer is loaded first.
SWAP	an asset is swapped in the viewer using <code>setAsset()</code> API.
PLAY	playback is started.
PAUSE	playback is paused.
STOP	playback is stopped.
MILESTONE	playback reaches one of the following millstones: 0%, 25%, 50%, 75%, and 100%.

HTTP video delivery



Note: Secure Video Delivery only applies to AEM 6.2 with the installation of [Feature Pack-13480](#) and to AEM 6.1 with installation of [Feature Pack NPR-15011](#).

Provided that the viewer works in configuration as outlined at the beginning of this section, published video delivery can happen both in HTTPS (secure) and HTTP (insecure) modes. In a default configuration, the video delivery protocol strictly follows the delivery protocol of the embedding web page. However, it is possible to force HTTPS video delivery without regard to the protocol used by embedding the web page using the [VideoPlayer.ssl](#) configuration attribute. (Note that video preview in Author mode is always delivered securely over HTTPS.)

Depending on the method of publishing Dynamic Media video that you use in AEM, the `VideoPlayer.ssl` configuration attribute is applied differently as demonstrated in the following:

- If you publish a Dynamic Media video with a URL, you append `VideoPlayer.ssl` to the URL. For example, to force secure video delivery, you append `&VideoPlayer.ssl=on` to the end of the following viewer URL example:

~~<https://demos-pub.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/VideoViewer.js?VideoPlayer.ssl=on>~~

See also [\(AEM 6.2\) Linking URLs to your Web Application](#) or [\(AEM 6.1\) Linking URLs to your Web Application](#)

- If you publish a Dynamic Media video with embed code, you add `VideoPlayer.ssl` to the list of other viewer configuration parameters in the embed code snippet. For example, to force HTTPS video delivery, you append `&VideoPlayer.ssl=on` as in the following example:

```
<style type="text/css">
  #s7video_div.s7videoviewer{
    width:100%;
    height:auto;
  }
</style>
<script type="text/javascript"
src="https://demos-pub.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/VideoViewer.js"></script>
<div id="s7video_div"></div>
<script type="text/javascript">
  var s7videoviewer = new s7viewers.VideoViewer({
    "containerId" : "s7video_div",
    "params" : {
      "VideoPlayer.ssl" : "on",
      "serverurl" : "https://adobedemo62-h.assetsadobe.com/is/image",
      "contenturl" : "https://demos-pub.assetsadobe.com/",
      "config" : "/etc/dam/presets/viewer/Video",
      "config2" : "/etc/dam/presets/analytics",
      "videoseverurl": "https://gateway-na.assetsadobe.com/DMGateway/public/demoCo",
      "posterimage":
"/content/dam/marketing/shoppable-video/adobe-axis-demo/Adobe_AXIS_V3_GRADED-HD.mp4",
      "asset" :
"/content/dam/marketing/shoppable-video/adobe-axis-demo/Adobe_AXIS_V3_GRADED-HD.mp4" }
  }).init();
</script>
```

See also [\(AEM 6.2\) Embedding the Video on a Web Page](#) or [\(AEM 6.1\) Embedding the Video on a Web Page](#).

Localization of user interface elements

Certain content that the Video Viewer displays is subject to localization. This content includes user interface element tool tips and an error message that is displayed when the video cannot be played.

Every textual content in the viewer that can be localized is represented by a special Viewer SDK identifier called SYMBOL. Any SYMBOL has a default associated text value for the English locale ("en") supplied with the out-of-the-box viewer. It also may have user-defined values set for as many locales as needed.

When the viewer starts, it checks the current locale to see if there is a user-defined value for each supported SYMBOL for the locale. If there is, it uses the user-defined value; otherwise, it falls back to the out-of-the-box default text.

User-defined localization data can be passed to the viewer as a localization JSON object. Such object contains the list of supported locales, SYMBOL text values for each locale, and the default locale.

An example of such a localization object is the following:

```
{
  "en":{
    "VideoPlayer.ERROR":"Your Browser does not support HTML5 Video tag or the video cannot be
played.",
  },
}
```

```

"PlayPauseButton.TOOLTIP_SELECTED": "Play"
},
"fr": {
"VideoPlayer.ERROR": "Votre navigateur ne prend pas en charge la vidéo HTML5 tag ou la vidéo ne
peuvent pas être lus.",
"PlayPauseButton.TOOLTIP_SELECTED": "Jouer"
},
defaultLocale: "en"
}

```

In the above example, the localization object defines two locales ("en" and "fr") and provides localization for two user interface elements in each locale.

The web page code should pass such localization object to the viewer constructor as a value of `localizedTexts` field of the configuration object. An alternative option is to pass the localization object by calling the `setLocalizedTexts(localizationInfo)` method.

The following SYMBOLs are supported:

SYMBOL	Description
<code>PlayPauseButton.TOOLTIP_SELECTED</code>	Tool tip for the selected play pause button state.
<code>PlayPauseButton.TOOLTIP_UNSELECTED</code>	Tool tip for the deselected play pause button state.
<code>PlayPauseButton.TOOLTIP_REPLAY</code>	Tool tip for the play pause button state.
<code>VideoScrubber.TOOLTIP</code>	Tool tip for the video scrubber.
<code>VideoTime.TOOLTIP</code>	Tool tip for the video time on control bar.
<code>MutableVolume.TOOLTIP_SELECTED</code>	Tool tip for the selected mutable volume state.
<code>MutableVolume.TOOLTIP_UNSELECTED</code>	Tool tip for the deselected mutable volume.
<code>FullScreenButton.TOOLTIP_SELECTED</code>	Tool tip for the selected full screen button state.
<code>FullScreenButton.TOOLTIP_UNSELECTED</code>	Tool tip for the deselected full screen button state.
<code>ClosedCaptionButton.TOOLTIP_SELECTED</code>	Tool tip for the selected closed caption button state.
<code>ClosedCaptionButton.TOOLTIP_UNSELECTED</code>	Tool tip for the deselected closed caption button state.
<code>SocialShare.TOOLTIP</code>	Tool tip for the social share tool.
<code>EmailShare.TOOLTIP</code>	Tool tip for the email share button.
<code>EmailShare.HEADER</code>	Tool tip for the email dialog header.

SYMBOL	Description
EmailShare.TOOLTIP_HEADER_CLOSE	Tool tip for the email dialog box upper-right close button.
EmailShare.INVALID_ADDRESSES	Tool tip for the error message displayed in case email address is malformed.
EmailShare.TO	Label for the "To" input field.
EmailShare.TOOLTIP_ADD	Tool tip for the "Add Another Email Address" button.
EmailShare.ADD	Caption for "Add Another Email Address" button.
EmailShare.FROM	Label for the "From" input field.
EmailShare.MESSAGE	Label for the "Message" input field.
EmailShare.TOOLTIP_REMOVE	Tool tip for the "Remove Email Address" button.
EmailShare.CANCEL	Caption for the "Cancel" button.
EmailShare.TOOLTIP_CANCEL	Tool tip for the "Cancel" button.
EmailShare.CLOSE	Caption for the close button displayed in the bottom of dialog after form submission.
EmailShare.TOOLTIP_CLOSE	Tool tip for the close button displayed in the bottom of dialog after form submission.
EmailShare.ACTION	Caption for the form submission button.
EmailShare.TOOLTIP_ACTION	Tool tip for the form submission button.
EmailShare.SEND_SUCCESS	Confirmation message displayed when email was sent successfully.
EmailShare.SEND_FAILURE	Error message that is displayed when email was not sent successfully.
EmbedShare.TOOLTIP	Tool tip for the embed share button.
EmbedShare.HEADER	Tool tip for the embed dialog box header.
EmbedShare.TOOLTIP_HEADER_CLOSE	Tool tip for the embed dialog box upper-right close button.

SYMBOL	Description
EmbedShare.DESRIPTION	Description of the embed code text.
EmbedShare.EMBED_SIZE	Label for the embed size combo box.
EmbedShare.CANCEL	Caption for the "Cancel" button.
EmbedShare.TOOLTIP_CANCEL	Tool tip for the "Cancel" button.
EmbedShare.ACTION	Caption for "Select All" button.
EmbedShare.TOOLTIP ACTION	Tooltip for "Select All" button.
EmbedShare.CUSTOM_SIZE	Text for the last "custom size" entry in the embed size combo box.
LinkShare.TOOLTIP	Tool tip for the link share button.
LinkShare.HEADER	Tool tip for the link dialog box header.
LinkShare.TOOLTIP_HEADER_CLOSE	Tool tip for the link dialog box upper-right close button.
LinkShare.DESRIPTION	Description of the share link.
LinkShare.CANCEL	Caption for the "Cancel" button.
LinkShare.TOOLTIP_CANCEL	Tool tip for the "Cancel" button.
LinkShare.ACTION	Caption for "Select All" button.
LinkShare.TOOLTIP ACTION	Tooltip for "Select All" button.
FacebookShare.TOOLTIP	Tool tip for the Facebook share button.
TwitterShare.TOOLTIP	Tool tip for the Twitter share button.
VideoPlayer.ERROR	Tool tip for the error message that appears when no video playback is possible.

Full screen support

The viewer supports full screen operation mode.

On modern desktop browsers, except Internet Explorer 10 and older, and on some touch devices, the viewer uses "native" full screen mode. This mode means that the entire device screen is occupied by the viewer content.

On iOS devices and on older Internet Explorer browsers, the viewer uses "simulated" full screen mode instead. In this mode, the viewer simply resizes to take the full area of the web browser window. Also, the web browser Chrome and other windows are still visible on the screen.

An end user enters and leaves full screen mode by pressing the Full Screen button in the viewer user interface. When "native" full screen mode is used on desktop, it is also possible to exit it by pressing **Esc**.

External video support

The viewer supports playing video hosted outside of Scene7 Publishing System or AEM Dynamic Media.

Supported formats for the external video are either MP4 in H.264 format or M3U8 manifest for HLS stream.

The viewer can work either Scene7 or Dynamic Media video or with external video. If the viewer starts with Scene7/Dynamic Media video, use it with such asset type going forward, it is not possible to load an external video into this viewer using `method`. And vice versa: if viewer was initially loaded with external video, it should keep working with external videos only.

When working with external video the viewer ignores the value of playback modifier and detects the playback type from the external video extension. If external video URL ends with `.m3u8` the viewer is using HLS playback, otherwise progressive playback is used.

Viewer SDK namespace

The viewer is built of many Viewer SDK components. In most cases, the web page does not need to interact with SDK components API directly; all common needs are covered in the viewer API itself.

However, some advanced use cases require that the web page obtain a reference to an inner SDK component using the `getComponent()` viewer API and then use all the flexibility of the APIs of SDK itself.

The namespace that is used to load and initialize SDK components by the viewer depends on the environment in which the viewer is operating. If the viewer is running in AEM (Adobe Experience Manager), the viewer loads SDK components into `s7viewers.s7sdk` namespace. Likewise, the viewer served from Scene7 Publishing System loads the SDK into `s7classic.s7sdk`.

In either case, the namespace used by the SDK inside the viewer has either `s7viewers` or `s7classic` as the prefix. And, it is different from the plain `s7sdk` namespace used in the SDK User Guide or SDK API documentation. For that reason, it is important to use a fully qualified SDK namespace when you write custom application code that communicates with internal viewer components.

For example, if you plan to listen to `StatusEvent.NOTF_VIEW_READY` event and the viewer is served from AEM, the fully qualified event type is `s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY`, and the event listener code looks similar to the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var videoPlayer = <instance>.getComponent("videoPlayer");
    videoPlayer.addEventListener(s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e)
    {
      console.log("view ready");
    }, false);
  }
});
```

The same code for the viewer served from Scene7 Publishing System looks like the following:

```
<instance>.setHandlers({
```

```
"initComplete":function() {
  var videoPlayer = <instance>.getComponent("videoPlayer");
  videoPlayer.addEventListener(s7classic.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e)
  {
    console.log("view ready");
  }, false);
});
```

Zoom

Zoom Viewer is an image viewer that displays a zoomable image. This viewer works with image sets and navigation is done by using swatches. It has zoom tools, full screen support, swatches, and an optional close button. It is designed to work on desktops and mobile devices.



Note: Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.

Viewer type 502.

See [System requirements](#).

Demo URL

<https://s7d9.scene7.com/s7viewers/html5/ZoomViewer.html?asset=Scene7SharedAssets/ImageSet-Views-Sample>

Using Zoom Viewer

Zoom Viewer represents a main JavaScript file and a set of helper files (a single JavaScript include with all Viewer SDK components used by this particular viewer, assets, CSS) downloaded by the viewer in runtime.

You can use Zoom Viewer in pop-up mode using a production-ready HTML page provided with IS-Viewers or in embedded mode, where it is integrated into target web page using documented API.

Configuration and skinning are similar to that of the other viewers. All skinning is achieved by way of custom CSS.

See [Command reference common to all viewers – Configuration attributes](#) and [Command reference common to all Viewers – URL](#)

Interacting with Zoom Viewer

Zoom Viewer supports the following touch gestures that are common in other mobile applications. When the viewer cannot process user's swipe gesture it forwards the event to the web browser to perform native page scroll. This kind of functionality lets the user navigate through the page even if the viewer occupies most of the device screen area.

Gesture	Description
Single tap	Selects new thumbnail in swatches. In the main view, it hides or reveals user interface elements.
Double tap	Zooms in one level until maximum magnification is reached. The next double tap gesture resets the viewer to the initial viewing state.
Pinch	Zooms in or out.

Gesture	Description
Horizontal swipe or flick	<p>Scrolls through the list of swatches in the swatch bar.</p> <p>If the image is in a reset state and the <code>frametransition</code> parameter is set to <code>slide</code>, the asset is changed with the slide animation. For other <code>frametransition</code> modes the gesture performs native page scrolling.</p> <p>If the image is zoomed in, it moves the image horizontally. If image is moved to the view edge and a swipe is performed in the same direction, the gesture performs native page scrolling.</p>
Vertical swipe	<p>If the image is in a reset state, the gesture performs native page scrolling.</p> <p>When the image is zoomed in, it moves the image vertically. If the image is moved to the view edge and a swipe is performed in the same direction, the gesture performs native page scroll.</p> <p>If the gesture is done within the swatches area, it performs a native page scroll.</p>

The viewer supports both touch input and mouse input on Windows devices with a touch screen and mouse. This support, however, is limited to Chrome, Internet Explorer 11, and Edge web browsers only.

This viewer is fully keyboard accessible.

See [Keyboard accessibility and navigation](#).

Embedding Zoom Viewer

Different web pages have different needs for viewer behavior. Sometimes a web page provides a link that, when clicked, opens the viewer in a separate browser window. In other cases, it is necessary to embed the viewer directly in the hosting page. In the latter case, the web page may have static layout, or use responsive design that displays differently on different devices or for different browser window sizes. To accommodate these needs, the viewer supports three primary operation modes: pop-up, fixed size embedding, and responsive design embedding.

About pop-up mode

In pop-up mode, the viewer is opened in a separate web browser window or tab. It takes the entire browser window area and adjusts in case the browser is resized or the device orientation is changed.

This mode is the most common for mobile devices. The web page loads the viewer using `window.open()` JavaScript call, properly configured A HTML element, or any other suitable method.

It is recommended that you use an out-of-the-box HTML page for the pop-up operation mode. The out-of-the-box HTML page is called `ZoomViewer.html` and it is located under the `html5/` subfolder of your standard IS-Viewers deployment as in the following:

```
<s7viewers_root>/html5/ZoomViewer.html
```

Apply custom CSS to achieve a customized appearance for the page.

The following is an example of HTML code that opens the viewer in the new window:

```
<a
href="http://s7d1.scene7.com/s7viewers/html5/ZoomViewer.html?asset=Scene7SharedAssets/ImageSet-Views-Sample"
target="_blank">Open popup viewer</a>
```

About fixed size embedding mode and responsive design embedding mode

In the embedded mode the viewer is added to the existing web page, which may already have some customer content not related to the viewer. The viewer normally occupies only a part of the web page's real estate.

The primary use cases are web pages oriented for desktops or tablet devices, and also responsive designed pages that adjust layout automatically, depending on the device type.

Fixed size embedding is used when the viewer does not change its size after initial load. This option is the best choice for web pages that have a static layout.

Responsive design embedding mode assumes that the resizing of the viewer is necessary during runtime due to the size change of its container `DIV`. The most common use case is adding a viewer to a web page that uses a flexible layout.

In responsive design embedding mode, the viewer behaves differently depending on the way web page sizes its container `DIV`. If the web page only sets the width of the container `DIV`, leaving its height unrestricted, the viewer automatically chooses its height according to the aspect ratio of the asset that is used. This logic ensures that the asset fits perfectly into the view without any padding on the sides. This use case is the most common for web pages that use responsive layout frameworks such as Bootstrap, Foundation, and so forth.

If the web page sets both the width and the height for the viewer's container `DIV`, the viewer fills that area and follows the size that the web page provides. For example, embedding the viewer into a modal overlay, where the overlay is sized according to web browser window size.

Fixed size embedding

You add the viewer to a web page by doing the following:

1. Adding the viewer JavaScript file to your web page.
2. Defining the container `DIV`.
3. Setting the viewer size.
4. Creating and initializing the viewer.

1. Adding the viewer JavaScript file to your web page.

Creating a viewer requires that you add a script tag in the HTML head. Before you can use the viewer API, be sure that you include `ZoomViewer.js`. The `ZoomViewer.js` file is located under the `html5/js/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/html5/js/ZoomViewer.js
```

You can use a relative path if the viewer is deployed on one of the Adobe Scene7 servers and it is served from the same domain. Otherwise, you specify a full path to one of Adobe Scene7 servers that have the IS-Viewers installed.

The relative path looks like the following:

```
<script language="javascript" type="text/javascript"
src="/s7viewers/html5/js/ZoomViewer.js"></script>
```



Note: You should only reference the main viewer JavaScript `include` file on your page. You should not reference any additional JavaScript files in the web page code which might be downloaded by the viewer's logic in runtime. In particular, do not directly reference HTML5 SDK `Utils.js` library loaded by the viewer from `/s7viewers` context path (so-called consolidated SDK `include`). The reason is that the location of `Utils.js` or similar runtime viewer libraries is fully

managed by the viewer's logic and the location changes between viewer releases. Adobe does not keep older versions of secondary viewer includes on the server.

As a result, putting a direct reference to any secondary JavaScript include used by the viewer on the page breaks the viewer functionality in the future when a new product version is deployed.

2. Defining the container DIV.

Add an empty DIV element to the page where you want the viewer to appear. The DIV element must have its ID defined because this ID is passed later to the viewer API.

The placeholder DIV is a positioned element, meaning that the `position` CSS property is set to `relative` or `absolute`.

The following is an example of a defined placeholder DIV element:

```
<div id="s7viewer" style="position:relative"></div>
```

3. Setting the viewer size.

This viewer displays thumbnails when working with multi-item sets, on desktop systems thumbnails are placed below the main view. At the same time, the viewer allows the swapping of the main asset in runtime using `setAsset()` API. As a Developer, you have control over how the viewer manages the thumbnails area at the bottom when the new asset has only one item. It is possible to keep the outer viewer size intact and let the main view increase its height and occupy thumbnails area. Or, you can keep the main view size static and collapse the outer viewer area, letting web page content to move up, and use free screen real estate leftover from the thumbnails.

To keep the outer viewer bounds intact, define the size for the `.s7zoomviewer` top-level CSS class in absolute units. Sizing in CSS can be put right on the HTML page, or in a custom viewer CSS file, which is later assigned to a viewer preset record in Scene7 Publishing System or passed explicitly using a style command.

See [Customizing Zoom Viewer](#) for more information about styling the viewer with CSS.

The following is an example of defining a static outer viewer size in HTML page:

```
#s7viewer.s7zoomviewer {
  width: 640px;
  height: 480px;
}
```

You can see the behavior with a fixed outer viewer in the following example. Notice that when you switch between sets, the outer viewer size does not change:

https://marketing.adobe.com/resources/help/en_US/s7/viewers_ref/samples/ZoomViewer-fixed-outer-area.html

To make the main view dimensions static, define the viewer size in absolute units for the inner `Container` SDK component using the `.s7zoomviewer.s7container` CSS selector, or by using `stagesize` modifier.

The following is an example of defining the viewer size for the inner `Container` SDK component so that the main view area does not change its size when switching the asset:

```
#s7viewer.s7zoomviewer .s7container {
  width: 640px;
  height: 480px;
}
```

The following demo page shows the viewer behavior with a fixed main view size. Notice that when you switch between sets, the main view remains static and the web page content moves vertically.

https://marketing.adobe.com/resources/help/en_US/s7/viewers_ref/samples/ZoomViewer-fixed-main-view.html

You can set the `stagesize` modifier either in the viewer preset record in Scene7 Publishing System, or you can pass it explicitly with the viewer initialization code with the `params` collection or as an API call as described in the Command Reference section of this Help, as in the following:

```
zoomViewer.setParam( "stagesize",
"640,480" );
```

A CSS-based approach is recommended and is used in this example.

4. Creating and initializing the viewer.

When you have completed the steps above, you create an instance of `s7viewers.ZoomViewer` class, pass all configuration information to its constructor and call `init()` method on a viewer instance.

Configuration information is passed to the constructor as a JSON object. At minimum, this object should have `containerId` field which holds the name of the viewer container ID and nested `params` JSON object with configuration parameters that the viewer supports. In this case, the `params` object must have at least the Image Serving URL passed as `serverUrl` property and the initial asset as `asset` parameter. The JSON-based initialization API lets you create and start the viewer with a single line of code.

It is important to have the viewer container added to the DOM so that the viewer code can find the container element by its ID. Some browsers delay building DOM until the end of the web page. For maximum compatibility, call the `init()` method just before the closing BODY tag, or on the `body onload()` event.

At the same time, the container element should not necessarily be part of the web page layout just yet. For example, it may be hidden using `display:none` style assigned to it. In this case, the viewer delays its initialization process until the moment when the web page brings the container element back to the layout. When this occurs, the viewer load automatically resumes.

The following is an example of creating a viewer instance, passing the minimum necessary configuration options to the constructor, and calling the `init()` method. This example assumes `zoomViewer` is the viewer instance, `s7viewer` is the name of placeholder DIV, `http://s7d1.scene7.com/is/image/` is the Image Serving URL, and `Scene7SharedAssets/ImageSet-Views-Sample` is the asset.

```
<script type="text/javascript">
var zoomViewer = new s7viewers.ZoomViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
    "serverurl": "http://s7d1.scene7.com/is/image/"
  }
}).init();
</script>
```

The following code is a complete example of a trivial web page that embeds the Video Viewer with a fixed size:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7d1.scene7.com/s7viewers/html5/js/ZoomViewer.js"></script>
<style type="text/css">
#s7viewer.s7zoomviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var zoomViewer = new s7viewers.ZoomViewer({
  "containerId": "s7viewer",
```

```

"params": {
  "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
  "serverurl": "http://s7dl.scene7.com/is/image/"
}
}).init();
</script>
</body>
</html>

```

Responsive design embedding with unrestricted height

With responsive design embedding, the web page normally has some kind of flexible layout in place that dictates the runtime size of the viewer's container DIV. For the following example, assume that the web page allows the viewer's container DIV to take 40% of the web browser window size, leaving its height unrestricted. The web page HTML code would look like the following:

```

<!DOCTYPE html>
<html>
<head>
<style type="text/css">
.holder {
  width: 40%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>

```

Adding the viewer to such a page is similar to the steps for fixed size embedding. The only difference is that you do not need to explicitly define the viewer size.

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Creating and initializing the viewer.

All the steps above are the same as with the fixed size embedding. Add the container DIV to the existing "holder" DIV. The following code is a complete example. Notice how viewer size changes when the browser is resized, and how the viewer aspect ratio matches the asset.

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/ZoomViewer.js"></script>
<style type="text/css">
.holder {
  width: 40%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var zoomViewer = new s7viewers.ZoomViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  }
}).init();

```

```
</script>
</body>
</html>
```

The following examples page illustrates more real-life uses of responsive design embedding with unrestricted height:

https://marketing.adobe.com/resources/help/en_US/s7/vlist/vlist.html

Flexible-size embedding with width and height defined

In case of flexible-size embedding with width and height defined, the web page styling is different. It provides both sizes to the "holder" DIV and center it in the browser window. Also, the web page sets the size of the HTML and BODY element to 100 percent.

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>
```

The rest of the embedding steps are identical to the steps used for responsive design embedding with unrestricted height. The resulting example is the following:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7d1.scene7.com/s7viewers/html5/js/ZoomViewer.js"></script>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var zoomViewer = new s7viewers.ZoomViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"Scene7SharedAssets/ImageSet-Views-Sample",
```

```

    "serverurl": "http://s7dl.scene7.com/is/image/"
  }
}).init();
</script>
</body>
</html>

```

Embedding using setter-based API

Instead of using JSON-based initialization, it is possible to use setter-based API and no-args constructor. Using this API constructor does not take any parameters and configuration parameters are specified using `setContainerId()`, `setParam()`, and `setAsset()` API methods with separate JavaScript calls.

The following example illustrates using fixed size embedding with setter-based API:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://s7dl.scene7.com/s7viewers/html5/js/ZoomViewer.js"></script>
<style type="text/css">
#s7viewer.s7zoomviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var zoomViewer = new s7viewers.ZoomViewer();
zoomViewer.setContainerId("s7viewer");
zoomViewer.setParam("serverurl", "http://s7dl.scene7.com/is/image/");
zoomViewer.setAsset("Scene7SharedAssets/ImageSet-Views-Sample");
zoomViewer.init();
</script>
</body>
</html>

```

Command reference – Configuration attributes

Configuration attributes documentation for Zoom Viewer.

Any configuration command can be set in URL or using `setParam()`, or `setParams()`, or both, API methods. Any config attribute can be also specified in the server-side configuration record.

Some configuration commands may be prefixed with the class name or instance name of corresponding Viewer SDK component. An instance name of the component is dynamic and depends on the ID of the viewer container DOM element passed to `setContainerId()` API method. Documentation includes an optional prefix for such commands. For example, `zoomstep` command is documented as follows:

```
[ZoomView.|<containerId>_zoomView].zoomstep
```

which means that you can use this command as:

- `zoomstep` (short syntax)
- `ZoomView.zoomstep` (qualified with component class name)
- `cont_zoomView.zoomstep` (qualified with component ID, assuming `cont` is the ID of the container element)

See also [Command reference common to all viewers – Configuration attributes](#)

closebutton

closebutton=0|1

0-1	Set to 1 to enable close button display, or set to 0 to hide close button.
-----	--

Properties

Optional.

Default

0

Example

closebutton=1

Swatches.align

[Swatches.|<containerId>_swatches.]align=left|center|right,top|center|bottom

Specifies the internal alignment (anchoring) of the swatches container within the component area. In Swatches, the internal thumbnail container is sized so that only a whole number of swatches is shown. As a result, there is some padding between internal container and external component bounds. This command specifies how the internal swatches container is positioned inside component.

left center right	Sets the alignment of the horizontal swatches.
top center bottom	Sets the alignment of the vertical swatches.

Properties

Optional.

Default

center,center

Example

align=left,top

Swatches.buttonsnapmode

[Swatches.|<containerId>_swatches.]buttonsnapmode=snapin|snapout|overlay

<i>snapin</i>	Causes the buttons to align next to the swatches.
---------------	---

<i>snapout</i>	Causes the buttons to align next to the component border.
<i>overlay</i>	Causes the buttons to render on top of the swatches.

Properties

Optional.

Default

snapout

Example

```
buttonsnapmode=overlay
```

Swatches.direction

```
[Swatches.|<containerId>_swatches.]direction=auto|left|right
```

<i>auto left right</i>	<p>Specifies the way swatches fill in the view.</p> <p><i>left</i> sets left-to-right fill order;</p> <p><i>right</i> reverses the order so that the view is filled in from right-to-left, and top-to-bottom.</p> <p>When <i>auto</i> is set, the component applies <i>right</i> mode when locale is set to <i>ja</i>; otherwise, <i>left</i> is used.</p>
------------------------	--

Properties

Optional.

Default

auto

Example

```
direction=right
```

Swatches.enabledragging

```
[Swatches.|<containerId>_swatches.]enabledragging=0|1[,overdragvalue]
```

<i>0 1</i>	Enables or disables the ability for a user to scroll the swatches with a mouse or by using touch gestures
<i>overdragvalue</i>	Functions within the 0–1 range. It is a % value for movement in the wrong direction of the actual speed. If it is set to 1, it moves with the mouse. If it is set to 0, it does not let you move in the wrong direction at all.

Properties

Optional.

Default

1,0.5

Example

enabledragging=0

Swatches.fmt

[Swatches.|<containerId>_swatches.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha

jpg jpeg png png-alpha gif gif-alpha	Specifies the image format that the component uses for loading images from Image Server. If the specified format ends with -alpha, the component renders images as transparent content. For all other image formats the component treats images as opaque. Note that the component has a white background by default. Therefore, to make the background transparent set the background-color CSS property to transparent.
--------------------------------------	---

Properties

Optional.

Default


jpeg

Example

fmt-png-alpha

Swatches.iscommand

[Swatches.|<containerId>_swatches.]iscommand=isCommand

<i>isCommand</i>	<p>The Image Serving command string that is applied to all swatches. If it is specified in the URL, be sure that you HTTP-encode all occurrences of & and = as %26 and %3D, respectively.</p> <p> Note: Image sizing manipulation commands are not supported.</p>
------------------	---

Properties

Optional.

Default

None.

Example

When specified in the viewer URL.

```
iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000
```

When specified in the configuration data.

```
iscommand=op_sharpen=1&op_colorize=0xff0000
```

Swatches.maxloadradius

```
[Swatches.|<containerId>_swatches.]maxloadradius=-1|0|preloadnbr
```

-1 0 preloadnbr	<p>Specifies the component preload behavior. When set to -1 all swatches are loaded simultaneously when the component is initialized or the asset is changed.</p> <p>When set to 0 only visible swatches are loaded.</p> <p>preloadnbr defines how many invisible rows/columns around the visible area are preloaded.</p>
-----------------	---

Properties

Optional.

Default

1

Example

```
maxloadradius=-1
```

Swatches.pagemode

```
[Swatches.|<containerId>_swatches.]pagemode=0|1
```

0 1	<p>When toggled, the scroll buttons automatically cause the swatches to jump a full page length.</p> <p>Extra whitespace is shown on the last page if the swatches do not fit. Also, the last page has the same number of cells as any previous page.</p> <p>The scrollstep is ignored and mouse scrolling settles only on full pages.</p>
-----	--

Properties

Optional.

Default

0

Example

```
pagemode=1
```

Swatches.partialswatches

[Swatches.|<containerId>_swatches.]partialswatches=0|1

0 1	Specifies whether the component allows scrolling to stop when any of the swatches are partially visible (scrolling is not aligned). The recommended value is <code>false</code> or 0.
-----	---

Properties

Optional.

Default

0

Example

partialswatches=1

Swatches.scrollstep

[Swatches.|<containerId>_swatches.]scrollstep=*hStep*,*vStep*

<i>hStep</i>	Horizontal step.
<i>vStep</i>	Vertical step.

Specifies the number of swatches to scroll for each click or tap of the corresponding scroll button.

Properties

Optional.

Default

3,3

Example

scrollstep=1,1

ZoomView.doubleclick

[ZoomView.|<containerId>_zoomView.]doubleclick=`none`|`zoom`|`reset`|`zoomReset`

<code>none</code> <code>zoom</code> <code>reset</code> <code>zoomReset</code>	Configures the mapping of double-click/tap to zoom actions. Setting to <code>none</code> disables double-click/tap zoom. If set to <code>zoom</code> clicking the image zooms in one zoom step; CTRL+Click zooms out one zoom step. Setting to <code>reset</code> causes a single click on the image to reset the
---	---

	zoom to the initial zoom level. For <code>zoomReset</code> , reset is applied if the current zoom factor is at or beyond the specified limit, otherwise zoom is applied.
--	--

Properties

Optional.

Default

`reset` on desktop computers; `zoomReset` on touch devices.

Example

`doubleclick=zoom`

ZoomView.enableHD

`[ZoomView.<containerId>_zoomView.enableHD=always|never|limit[,number]`

<code>always never limit</code>	Enable, limit or disable optimization for devices where <code>devicePixelRatio</code> is greater than 1, that is devices with high-density display like iPhone4 and similar devices. If active then the component limits the size of the IS image request as if the device only had a pixel ratio of 1 and that way reducing the bandwidth. See example below.
<code>number</code>	If using the limit setting, the component enables high pixel density only up to the specified limit. See example below.

Properties

Optional.

Default

`limit,1500`

Example

The following are the expected results when you use this configuration attribute with the viewer, and the viewer size is 1000 x 1000:

If the set variable equals	Result
<code>always</code>	The pixel density of the screen/device is always taken into account. <ul style="list-style-type: none">• If the screen pixel density = 1, then the requested image is 1000 x 1000.• If the screen pixel density = 1.5, then the requested image is 1500 x 1500.

If the set variable equals	Result
	<ul style="list-style-type: none"> • If the screen pixel density = 2, then the requested image is 2000 x 2000.
never	It always uses a pixel density of 1 and ignores the device's HD capability. Therefore, the requested image requested is always 1000 x 1000.
limit<number>	<p>A device pixel density is requested and served only if the resulting image is below the specified limit.</p> <p>The limit number applies to either the width or the height dimension.</p> <ul style="list-style-type: none"> • If the limit number is 1600, and the pixel density is 1.5, then the 1500 x 1500 image is served. • If the limit number is 1600, and the pixel density is 2, then the 1000 x 1000 image is served because the 2000 x 2000 image exceeds the limit. <p>Best practice: The limit number needs to work in conjunction with the company setting for maximum size image. Therefore, set the limit number to equal the company maximum image size setting.</p>

ZoomView.fmt

[ZoomView.|<containerId>_zoomView.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha

jpg jpeg png png-alpha gif gif-alpha	Specifies the image format that the component uses for loading images from Image Server. If the specified format ends with -alpha, the component renders images as transparent content. For all other image formats the component treats images as opaque. The component has a white background by default. Therefore, to make it transparent, set the background-color CSS property to transparent.
--------------------------------------	--

Properties

Optional.

Default

jpeg

Example

fmt=png-alpha

ZoomView.frametransition

[ZoomView.|<containerId>_zoomView.]frametransition=none|fade|slide[,duration[,spacing]]

<code>none</code> <code>fade</code> <code>slide</code>	Specifies the type of the effect applied on frame change. <code>none</code> stands for no transition; frame change happens instantly. <code>fade</code> means cross-fade transition between old and new frames. <code>slide</code> activates transition where the old frame slides out of the view and the new frame slides in.
<code>duration</code>	Specifies the duration (in second) of <code>fade</code> or <code>slide</code> transition effect.
<code>spacing</code>	The spacing between adjacent frames in <code>slide</code> transition, has the range between 0 and 1 and is relative to component's width.

Properties

Optional.

Default

None.

Example

```
frametransition=fade,1
```

ZoomView.iconeffect

```
[ZoomView. | <containerId>_zoomView.] iconeffect=0 | 1[,count][,fade][,autoHide]
```

<code>0</code> <code>1</code>	Enables the <code>iconeffect</code> to display on the top of the image when the image is in a reset state and it is suggestive of an available action to interact with the image.
<code>count</code>	Specifies the maximum number of times the <code>iconeffect</code> appears and reappears. A value of <code>-1</code> indicates that the icon always reappears indefinitely.
<code>fade</code>	Specifies the duration of the show or hide animation, in seconds.
<code>autoHide</code>	Sets the number of seconds that the <code>iconeffect</code> stays fully visible before it auto hides. That is, the time after the fade-in animation is completed but before the fade out animation starts. A setting of <code>0</code> disables the auto-hide behavior.

Properties

Optional.

Default


```
1,1,0.3,3
```

Example

```
iconeffect=0
```

ZoomView.iscommand

```
[ZoomView. | <containerId>_zoomView. ]iscommand=isCommand
```

<i>iscommand</i>	<p>The Image Serving command string that is applied to zoom image. If specified in the URL all occurrences of & and = must be HTTP-encoded as %26 and %3D, respectively.</p> <p> Note: Image sizing manipulation commands are not supported.</p>
------------------	--

Properties

Optional.

Default

None.

Example

When specified in the viewer URL:

```
iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000
```

When specified in the config data:

```
iscommand=op_sharpen=1&op_colorize=0xff0000
```

ZoomView.reset

```
[ZoomView. | <containerId>_zoomView. ]reset=0|1
```

0 1	<p>Resets the view port when frame (image) changes. If set to 0 it preserves the current view port with the best possible fit while preserving the aspect ratio of the newly set image.</p>
-----	---

Properties

Optional.

Default

1

Example

```
reset=0
```

ZoomView.singleclick

```
[ZoomView. | <containerId>_zoomView. ]singleclick=none|zoom|reset|zoomReset
```

none zoom reset zoomReset	Configures the mapping of single-click/tap to zoom actions. Setting to <code>none</code> disables single-click/tap zoom. If set to <code>zoom</code> clicking the image zooms in one zoom step; CTRL+Click zooms out one zoom step. Setting to <code>reset</code> causes a single click on the image to reset the zoom to the initial zoom level. For <code>zoomReset</code> , reset is applied if the current zoom factor is at or beyond the specified limit, otherwise zoom is applied.
---------------------------------	--

Properties

Optional.

Default

zoomReset on desktop computers; none on touch devices.

Example

```
singleclick=zoom
```

ZoomView.transition

```
[ZoomView.<containerId>_zoomView.]transition=time[,easing]
```

<i>time</i>	Specifies the time in seconds that the animation for a single zoom step action takes.
<i>easing</i>	<p>Creates an illusion of acceleration or deceleration which makes the transition appear more natural. You can set easing to one of the following:</p> <ul style="list-style-type: none"> • 0 (auto) • 1 (linear) • 2 (quadratic) • 3 (cubic) • 4 (quartic) • 5 (quintic) <p>Auto mode always uses linear transition when elastic zoom is disabled (default). Otherwise, it fits one of the other easing functions based on the transition time. That is, the shorter the transition time the higher the easing function is used to accelerate the acceleration or deceleration effect.</p>

Properties

Optional.

Default

0.5,0

Example

```
transition=2,2
```

ZoomView.zoomstep

[ZoomView. | <containerId>_zoomView.]zoomstep=step[, limit]

<i>step</i>	Configures the number zoom in and zoom out actions that are required to increase or decrease the resolution by a factor of two. The resolution change for each zoom action is 2 ¹ per step. Set to 0 to zoom to full resolution with a single zoom action.
<i>limit</i>	Specifies the maximum zoom resolution, relative to the full resolution image. The default is 1.0, which does not allow zooming beyond full resolution.

Properties

Optional.

Default

1,1

Example

zoomstep=2,3

Javascript API reference for Zoom Viewer

The main class of the Zoom Viewer is BasicZoomViewer. It is declared in the s7viewers namespace. This JavaScript API covers constructor, methods, and call backs of this particular class.

In all the following examples, <instance> stands for the actual name of the JavaScript viewer object that is instantiated from the s7viewers.ZoomViewer class.

dispose

JavaScript API reference for Zoom Viewer.

dispose()

Disposes this viewer instance by releasing all resources used by the viewer logic and deleting all inner objects and components created by the viewer in runtime.

The web page code should also delete the viewer instance variable as well to completely remove the viewer from the web browser memory.

If the web page code has registered event listeners directly on Viewer SDK components used by the viewer—or stored external references to such components—such listeners must be explicitly unregistered by the web page code, and such external component references must be deleted prior to calling dispose().

Do not access the Viewer API any more after dispose() is called.

Parameters

None.

Returns

None.

Example

```
<instance>.dispose()
```

getComponent

JavaScript API reference for Video Viewer

```
getComponent(componentId)
```

Returns a reference to the Viewer SDK component that is used by the viewer. The web page can use this method to extend or customize the behavior of the out-of-box viewer. Call this method only after the `initComplete` viewer callback has run, otherwise the component may not be created yet by the viewer logic.

Parameters

componentID - {String} an ID of the Viewer SDK component used by the viewer. This viewer supports the following component IDs:

Component ID	Viewer SDK component class name
parameterManager	s7sdk.ParameterManager
container	s7sdk.common.Container
mediaSet	s7sdk.set.MediaSet
zoomView	s7sdk.image.ZoomView
swatches	s7sdk.set.Swatches
setIndicator	s7sdk.set.SetIndicator
zoomInButton	s7sdk.common.ZoomInButton
zoomOutButton	s7sdk.common.ZoomOutButton
zoomResetButton	s7sdk.common.ZoomResetButton
fullScreenButton	s7sdk.common.FullScreenButton
closeButton	s7sdk.common.CloseButton

When working with SDK APIs it is important to use correct fully qualified SDK namespace as described in [Viewer SDK namespace](#).

Refer to the Viewer SDK API documentation for more information about a particular component.

Returns

{Object} a reference to Viewer SDK component. The method returns null if the `componentId` is not a supported viewer component or if the component was not yet created by the viewer logic.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    var zoomView = <instance>.getComponent("zoomView");
  }
})
```

init

JavaScript API reference for Video Viewer.

```
init()
```

Starts the initialization of the Video Viewer. By this time the container DOM element must be created so that the viewer code can find it by its ID.

If the container element is not a part of the web page layout just yet (for example, it may be hidden using `display:none` style assigned to it, the viewer suspends its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load automatically resumes.

Call this method only once during viewer life cycle; subsequent calls are ignored.

Parameters

None.

Returns

{Object} A reference to the viewer instance.

Example

```
<instance>.init()
```

setAsset

JavaScript API reference for Video Viewer.

```
setAsset(asset)
```

<i>asset</i>	<p>{String} new asset id, explicit image set or explicit image set with frame-specific Image Serving modifiers, with optional global Image Serving modifiers appended after "?".</p> <p>Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.</p>
--------------	---

Sets the new asset. You can call this parameter at any time, either before or after `init()`. If it is called after `init()`, the viewer swaps the asset at runtime.

See also [init](#).

Returns

None.

Example

Single image reference:

```
<instance>.setAsset("Scene7SharedAssets/Backpack_B")
```

Single reference to an image set that is defined in a catalog:

```
<instance>.setAsset("Scene7SharedAssets/ImageSet-Views-Sample")
```

Explicit image set:

```
<instance>.setAsset("Scene7SharedAssets/Backpack_B,Scene7SharedAssets/Backpack_C")
```

Explicit image set with frame-specific Image Serving modifiers:

```
<instance>.setAsset("(Scene7SharedAssets/Backpack_B?op_colorize=255%20%20,Scene7SharedAssets/Backpack_B?op_colorize=0x00ff00)")
```

Sharpening modifier added to all images in the set:

```
<instance>.setAsset("Scene7SharedAssets/ImageSet-Views-Sample?op_sharpen=1")
```

setContainerId

JavaScript API reference for Video Viewer.

```
setContainerId(containerId)
```

Sets the ID of the DOM container (normally a DIV) into which the viewer is inserted. It is not necessary to have the container element created by the time this method is called. However, the container must exist when `init()` is run. It must be called before `init()`.

This method is optional if the viewer configuration information was passed with `config` JSON object to constructor.

<i>containerId</i>	{string} ID of container.
--------------------	---------------------------

Returns

None.

Example

```
<instance>.setContainerId("s7viewer");
```

setHandlers

JavaScript API reference for Video Viewer

```
setHandlers(handlers)
```

Specifies zero or more callback handlers. A call to this method fully overwrites event handlers that were previously assigned for that viewer instance. Must be called before `init()`.

Parameter

<i>handlers</i>	<p>{Object} JSON object with viewer event callbacks, where the property name is the name of the supported viewer event and the property value is a JavaScript function reference to an appropriate callback.</p> <p>See Event callbacks for more information about viewer events.</p>
-----------------	---

Returns

None.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    console.log("init complete");
  }
})
```

setLocalizedTexts

JavaScript API reference for Video Viewer.

setLocalizedTexts(*localizationInfo*)

<i>localizationInfo</i>	<p>{Object} JSON object with localization data.</p> <p>See Localization of user interface elements for more information.</p> <p>See also <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
-------------------------	--

Sets localization SYMBOL values for one or more locales. This parameter must be called before `init()`.

See also [init](#).

Returns

None.

Example

```
<instance>.setLocalizedTexts({"en":{"CloseButton.TOOLTIP":"Close"},"fr":{"CloseButton.TOOLTIP":"Fermer"},defaultLocale:"en"})
```

setParam

JavaScript API reference for Video Viewer.

setParam(*name*, *value*)

<i>name</i>	{string} name of parameter.
<i>value</i>	{string} value of parameter. The value cannot be percent-encoded.

Sets the viewer parameter to a specified value. The parameter is either a viewer-specific configuration option or a software development kit modifier. This parameter is called before `init()`.

This method is optional if the viewer configuration information was passed with `config` JSON object to constructor.

See also [init](#).

Returns

None.

Example

```
<instance>.setParam("style", "customStyle.css")
```

setParams

JavaScript API reference for Video Viewer.

`setParams(params)`

<i>params</i>	{string} name=value parameter pairs separated with &.
---------------	---

Sets one or more parameters to a given value. The method argument syntax is identical to a URL query string. That is, it represents name=value pairs separated with &. Just as in a query string, the names and values are percent-encoded using UTF8. Before you call `init()`, this parameter must be called.

This method is optional if the viewer configuration information was passed with `config` JSON object to constructor.

See also [init](#).

Returns

None.

Example

```
<instance>.setParams("ZoomView.zoomfactor=2,3&ZoomView.iscommand=op_sharpen%3d1")
```

ZoomViewer

JavaScript API reference for Zoom Viewer.

`ZoomViewer([config])`

Constructor, creates a new Zoom Viewer instance.

Parameters

<i>config</i>	<p>{object} optional JSON configuration object, allows all the viewer settings to pass to the constructor to avoid calling individual setter methods. Contains the following properties:</p> <ul style="list-style-type: none">• <code>containerId</code> - {String} ID of the DOM container (normally a <code>DIV</code>) that the viewer is inserted into. By the time this method is called, it is not necessary to have the container element created. However, the container must exist when <code>init()</code> is run. Required.
---------------	---

- `params` – {Object} JSON object with viewer configuration parameters where the property name is either viewer-specific configuration option or SDK modifier, and the value of that property is a corresponding settings value. Required.
- `handlers` – {Object} JSON object with viewer event callbacks, where the property name is the name of supported viewer event and the property value is a JavaScript function reference to appropriate callback. Optional.

See [Event callbacks](#) for more information about viewer events.

- `localizedTexts` – {Object} JSON object with localization data. Optional.

See [Localization of user interface elements](#) for more information.

See also *Viewer SDK User Guide* and the example for more information about the object's content.

Returns

None.

Example

```
var zoomViewer = new s7viewers.ZoomViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  },
  "handlers": {
    "initComplete": function() {
      console.log("init complete");
    }
  },
  "localizedTexts": {
    "en": {
      "CloseButton.TOOLTIP": "Close"
    },
    "fr": {
      "CloseButton.TOOLTIP": "Fermer"
    },
    defaultLocale: "en"
  }
});
```

Event callbacks

The viewer supports JavaScript event callbacks that the web page uses to track the viewer initialization process or runtime behavior.

Callback handlers are assigned by passing event names and corresponding handler functions with the `handlers` property to config JSON object in the viewer's constructor. Alternatively, it is possible to use `setHandlers()` API method.

Supported viewer events include the following:

- `initComplete` – triggers when viewer initialization is complete and all internal components are created, so that it is possible to use `getComponent()` API. The callback handler does not take any arguments.

- `trackEvent` – triggers each time an event occurs inside the viewer which may be handled by an event tracking system, such as Adobe Analytics. The callback handler takes the following arguments:
 - `objID` {String} not currently used.
 - `compClass` {String} not currently used.
 - `instName` {String} an instance name of the Viewer SDK component that triggered the event.
 - `timeStamp` {Number} event time stamp.
 - `eventInfo` {String} event payload.

See also [ZoomViewer](#) and [setHandlers](#).

Customizing Zoom Viewer

All visual customization and most behavior customization for the Zoom Viewer is done by creating a custom CSS.

The suggested workflow is to take the default CSS file for the appropriate viewer, copy it to a different location, customize it, and specify the location of the customized file in the `style=` command.

Default CSS files can be found at the following location:

```
<s7_viewers_root>/html5/ZoomViewer_light.css
```

The viewer is supplied with two out-of-the-box CSS files, for "light" and "dark" color schemes. The "light" version is used by default, but you can switch to the "dark" version by using the following standard CSS:

```
<s7_viewers_root>/html5/ZoomViewer_dark.css
```

Custom CSS file must contain the same class declarations as the default one. If a class declaration is omitted, the viewer does not function properly because it does not provide built-in default styles for user interface elements.

Alternative way to provide custom CSS rules is to use embedded styles directly on the web page or in one of linked external CSS rules.

When creating custom CSS keep in mind that the viewer assigns `.s7zoomviewer` class to its container DOM element. If you are using external CSS file passed with `style=` command, use `.s7zoomviewer` class as parent class in descendant selector for your CSS rules. If you are doing embedded styles on the web page, additionally qualify this selector with an ID of the container DOM element as follows:

```
#<containerId>.s7zoomviewer
```

Building responsive designed CSS

It is possible to target different devices and embedding sizes in CSS to make your content display differently depending on a user's device or a particular web page layout. This includes, but is not limited to, different layouts, user interface element sizes, and artwork resolution.

The viewer supports two mechanisms of creating responsive designed CSS: CSS markers and standard CSS media queries. You can use these independently or together.

CSS markers

To assist in creating responsive designed CSS, the viewer supports CSS markers. These are special CSS classes that are dynamically assigned to the top-level viewer container element based on the run-time viewer size and the input type used on the current device.

The first group of CSS markers includes `.s7size_large`, `.s7size_medium`, and `.s7size_small` classes. They are applied based on the run-time area of the viewer container. If the viewer area is equal or bigger than the size of a common desktop monitor then `.s7size_large` is used; if the area is close to a common tablet device then `.s7size_medium` is assigned. For areas similar to mobile phone screens then `.s7size_small` is set. The primary purpose of these CSS markers is to create different user interface layouts for different screens and viewer sizes.

The second group of CSS Markers contains `.s7mouseinput` and `.s7touchinput`. `.s7touchinput` is set if the current device has touch input capabilities; otherwise, `.s7mouseinput` is used. These markers are mostly intended to create user interface input elements with different screen sizes for different input types, because normally touch input requires larger elements. In case the device has both mouse input and touch capabilities, `.s7touchinput` is set and the viewer renders a touch-friendly user interface.

The following sample CSS sets the zoom in button size to 28 x 28 pixels on systems with mouse input and to 56 x 56 pixels on touch devices. In addition, it hides the button completely if the viewer size gets very small:

```
.s7zoomviewer.s7mouseinput .s7zoominbutton {
    width:28px;
    height:28px;
}
.s7zoomviewer.s7touchinput .s7zoominbutton {
    width:56px;
    height:56px;
}
.s7zoomviewer.s7size_small .s7zoominbutton {
    visibility:hidden;
}
```

To target devices with different pixel density you need to use CSS media queries. The following media query block would contain CSS specific to high-density screens:

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5)
{
}
```

Using CSS markers is the most flexible way of building responsive designed CSS because it lets you target not only the device screen size but the actual viewer size, which is useful for responsive design layouts.

You can use the default viewer CSS file as an example of a CSS Markers approach.

CSS media queries

You can also accomplish device sensing by using pure CSS media queries. Everything enclosed within a given media query block is applied only when it is run on a corresponding device.

When applied to Mobile Viewers use four CSS media queries, defined in your CSS, in the order listed below:

1. Contains only rules specific for all touch devices.

```
@media only screen and (max-device-width:13.5in) and (max-device-
height:13.5in) and (max-device-width:799px),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in)
and (max-device-height:799px)
{
}
```

2. Contains only rules specific for tablets with high resolution screens.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px) and (-webkit-min-device-pixel-ratio:1.5),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px) and (-webkit-min-device-pixel-ratio:1.5)
{
}
```


3. Contains only rules specific for all mobile phones.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in)
{
}
```

4. Contains only rules specific for mobile phones with high resolution screens.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in) and
(-webkit-min-device-pixel-ratio: 1.5),
  only screen and (device-width:720px) and (device-height:1280px) and
(-webkit-device-pixel-ratio: 2),
  only screen and (device-width:1280px) and (device-height:720px) and
(-webkit-device-pixel-ratio: 2)
{
}
```

Using a media queries approach, you should organize CSS with device sensing as follows:

- First, the desktop-specific section defines all properties that are either desktop-specific or common to all screens.
- And second, the four media queries go in the order defined above and provide CSS rules that are specific for the corresponding device type.

There is no need to duplicate the entire viewer CSS in each media query. Only properties that are specific to given devices are redefined inside a media query.

CSS Sprites

Many viewer user interface elements are styled using bitmap artwork and have more than one distinct visual state. A good example is a button that normally has at least 3 different states: "up", "over", and "down". Each state requires its own bitmap artwork assigned.

With a classic approach to styling, the CSS would have a separate reference to individual image file on the server for each state of the user interface element. The following is a sample CSS for styling zoom-in button:

```
.s7zoomviewer.s7mouseinput .s7zoominbutton[state='up'] {
background-image:url(images/v2/ZoomInButton_dark_up.png);
}
.s7zoomviewer.s7mouseinput .s7zoominbutton[state='over'] {
background-image:url(images/v2/ZoomInButton_dark_over.png);
}
.s7zoomviewer.s7mouseinput .s7zoominbutton[state='down'] {
background-image:url(images/v2/ZoomInButton_dark_down.png);
}
.s7zoomviewer.s7mouseinput .s7zoominbutton[state='disabled'] {
background-image:url(images/v2/ZoomInButton_dark_disabled.png);
}
```

The drawback to this approach is that the end user experiences flickering or delayed user interface response when the element is interacted with for the first time. This action occurs because the image artwork for the new element state is not yet downloaded. Also, this approach may have a slight negative impact on performance because of an increase in the number of HTTP calls to the server.

CSS sprites is a different approach where image artwork for all element states is combined into a single PNG file called a "sprite". Such "sprite" has all visual states for the given element positioned one after another. When styling a user interface element with sprites the same sprite image is referenced for all different states in the CSS. Also, the `background-position` property is used for each state to specify which part of the "sprite" image is used. You can structure a "sprite" image in any suitable way. Viewers normally have it vertically stacked. Below is a "sprite"-based example of styling the same zoom-in button from above:

```
.s7zoomviewer .s7zoominbutton[state] {
background-image: url(images/v2/ZoomInButton_dark_sprite.png);
```

```

}
.s7zoomviewer.s7mouseinput .s7zoominbutton[state='up'] {
background-position: -84px -560px;
}
.s7zoomviewer.s7mouseinput .s7zoominbutton[state='over'] {
background-position: -56px -560px;
}
.s7zoomviewer.s7mouseinput .s7zoominbutton[state='down'] {
background-position: -28px -560px;
}
.s7zoomviewer.s7mouseinput .s7zoominbutton[state='disabled'] {
background-position: -0px -560px;
}

```

General styling notes and advice

- When customizing the viewer user interface with CSS the use of the `!IMPORTANT` rule is not supported to style viewer elements. In particular, `!IMPORTANT` rule should not be used to override any default or run-time styling provided by the viewer or Viewer SDK. The reason is that it may affect the behavior of proper components. Instead, you should use CSS selectors with the proper specificity to set CSS properties that are documented in this reference guide.
- All paths to external assets within CSS are resolved against the CSS location, not the viewer HTML page location. Be aware of this rule when you copy the default CSS to a different location. Either copy the default assets as well or update paths within the custom CSS.
- The preferred format for bitmap artwork is PNG.
- Bitmap artwork is assigned to user interface elements using the `background-image` property.
- The `width` and `height` properties of a user interface element define its logical size. The size of the bitmap passed to `background-image` does not affect logical size.
- To use the high pixel density of high-resolution screens like Retina, specify bitmap artwork twice as large as the logical user interface element size. Then, apply the `-webkit-background-size:contain` property to scale the background down to the logical user interface element size.
- To remove a button from the user interface, add `display:none` to its CSS class.
- You can use various formats for color value that CSS supports. If you need transparency, use the format `rgba(R,G,B,A)`. Otherwise, you can use the format `#RRGGBB`.

Common User Interface Elements

The following is user interface elements reference documentation that applies to Video Viewer:

Close button

Clicking or tapping this button closes the containing web page. This button only appears if the `closebutton` parameter is set to 1. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7zoomviewer .s7closebutton
```

CSS property	Description
top	Position from the top border, including padding.

CSS property	Description
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#).

Example – to set up a close button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7zoomviewer .s7closebutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7zoomviewer .s7closebutton [state='up'] {
background-image:url(images/v2/CloseButton_dark_up.png);
}
.s7zoomviewer .s7closebutton [state='over'] {
background-image:url(images/v2/CloseButton_dark_over.png);
}
.s7zoomviewer .s7closebutton [state='down'] {
background-image:url(images/v2/CloseButton_dark_down.png);
}
.s7zoomviewer .s7closebutton [state='disabled'] {
background-image:url(images/v2/CloseButton_dark_disabled.png);
}
```

Focus highlight

Input focus highlight displayed around focused viewer UI element is controlled with the CSS class selector.

CSS properties

The appearance is controlled with the following CSS class selector:

```
.s7zoomviewer *:focus
```

CSS property	Description
outline	Focus highlight style.

Example – to disable the default browser focus highlight for all viewer user interface elements, add the following CSS selector to viewer's style sheet:

```
.s7zoomviewer *:focus {
  outline: none;
}
```

Full screen button

Causes the viewer to enter or exit full screen mode when clicked by the user. This button does not display if the viewer works in popup mode and the system does not support native full screen. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7zoomviewer .s7fullscreenbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the "full screen" state and *selected='false'* corresponds to the "normal" state.

The button tool tip can be localized. See [Localization of user interface elements](#).

Example – to set up a full screen button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states when selected or not selected:

```
.s7zoomviewer .s7fullscreenbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7zoomviewer .s7fullscreenbutton [selected='false'][state='up'] {
background-image:url(images/enterFullBtn_up.png);
}
.s7zoomviewer .s7fullscreenbutton [selected='false'][state='over'] {
background-image:url(images/enterFullBtn_over.png);
}
.s7zoomviewer .s7fullscreenbutton [selected='false'][state='down'] {
background-image:url(images/enterFullBtn_down.png);
}
.s7zoomviewer .s7fullscreenbutton [selected='false'][state='disabled'] {
background-image:url(images/enterFullBtn_disabled.png);
}
.s7zoomviewer .s7fullscreenbutton [selected='true'][state='up'] {
background-image:url(images/exitFullBtn_up.png);
}
.s7zoomviewer .s7fullscreenbutton [selected='true'][state='over'] {
background-image:url(images/exitFullBtn_over.png);
}
.s7zoomviewer .s7fullscreenbutton [selected='true'][state='down'] {
background-image:url(images/exitFullBtn_down.png);
}
.s7zoomviewer .s7fullscreenbutton [selected='true'][state='disabled'] {
background-image:url(images/exitFullBtn_disabled.png); }
}
```

Icon effect


The zoom indicator is overlaid on the main view area. It is displayed when the image is in a reset state and it also depends on iconeffect parameter.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7zoomviewer .s7zoomview .s7iconeffect
```

CSS property	Description
background-image	Zoom indicator artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
width	Zoom indicator width.
height	Zoom indicator height.

 **Note:** Icon effect supports the `media-type` attribute selector, which you can use to apply different icon effects on different devices. In particular, `media-type='standard'` corresponds to desktop systems where mouse input is normally used and `media-type='multitouch'` corresponds to devices with touch input.

Example – to set up a 100 x 100 pixel zoom indicator with different art for desktop systems and touch devices.

```
.s7zoomviewer .s7zoomview .s7iconeffect {
  width: 100px;
  height: 100px;
}
.s7zoomviewer .s7zoomview .s7iconeffect[media-type='standard'] {
  background-image:url(images/v2/IconEffect_zoom.png);
}
.s7zoomviewer .s7zoomview .s7iconeffect[media-type='multitouch'] {
  background-image:url(images/v2/IconEffect_pinch.png);
}
```

Main viewer area

The main view area is the area occupied by the zoom image and swatches. It usually sets to fit the available device screen when no size is specified.

When working in embedded mode (when an explicit size is given to the main viewer area), the viewer automatically decreases the height of its main area by the height of Swatches component that is in working with the single image and therefore, swatches are not needed.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7zoomviewer
```

CSS property	Description
width	The width of the viewer.
height	The height of the viewer.
background-color	Background color in hexadecimal format.

Example – to set up a viewer with a white background (#FFFFFF) and make its size 512 x 288 pixels.

```
.s7zoomviewer {
  background-color: #FFFFFF;
  width: 512px;
  height: 288px;
}
```

Set indicator

Set indicator is a series of dots rendered on top of swatches when a viewer is used on a touch device. The dots help users to navigate through pages of thumbnails when scroll buttons are not available.

CSS properties of the set indicator

The appearance of the set indicator container is controlled with the following CSS class selector:

```
.s7zoomviewer .s7setindicator
```

CSS property	Description
background-color	The background color in hexadecimal format of the set indicator.

Example – to set up set indicator with a white background:

```
.s7zoomviewer .s7setindicator {
  background-color: #FFFFFF;
}
```

The appearance of an individual set indicator dot is controlled with the CSS class selector:

```
.s7zoomviewer .s7setindicator .s7dot
```

CSS property	Description
width	Width of the set indicator dot.
height	Height of the set indicator dot.
margin-left	Left margin in pixels.
margin-top	Top margin in pixels.
margin-right	Right margin in pixels.
margin-bottom	Bottom margin in pixels.
border-radius	Border radius in pixels.
background-color	Background color in hexadecimal format.



Note: Set indicator dot supports the *state* attribute selector, which can be used to apply different skins to different thumbnail states. In particular, *state="selected"* corresponds to the current page of thumbnails, *state="unselected"* corresponds to the default dot state.

Example – to set up set indicator dot to be 15 x 15 pixels, with two pixels horizontal margin, five pixels top margin, one pixel bottom margin, twelve pixels radius, #D5D3D3 default color, and #939393 active color:

```
.s7zoomviewer .s7setindicator .s7dot {
  width:15px;
  height:15px;
  margin-left:2px;
  margin-top:5px;
  margin-right:2px;
  margin-bottom:1px;
  border-radius:12px;
  background-color:#D5D3D3;
}
.s7zoomviewer .s7setindicator .s7dot[state="selected"] {
  background-color:#939393;
}
```

Swatches

Swatches consist of a row of thumbnail images with optional scroll buttons on the left and right hand side. Scroll buttons are only visible on the desktop if all thumbnails cannot fit into the container width. On mobile devices, or if thumbnails can fit into the container width, the scroll buttons are not shown.

```
.s7zoomviewer .s7swatches
```

CSS properties of the main viewer area

The appearance of the swatches container is controlled with the following CSS class selector:

```
.s7zoomviewer .s7zoomresetbutton
```

CSS property	Description
width	Width of the swatches.
height	Height of the swatches.
bottom	Vertical swatches offset relative to the viewer container.

Example – to set up swatches to 460 x 100 pixels.

```
.s7zoomviewer .s7swatches {
  width: 460px;
  height: 100px;
}
```

The spacing between the swatch thumbnails is controlled with the following CSS class selector:

```
.s7zoomviewer .s7swatches .s7thumbcell
```

CSS property	Description
margin	The size of the horizontal and vertical margin around each thumbnail. Actual thumbnail spacing is equal to the sum of the left and right margin that is set for .s7thumbcell.

Example

To set spacing to ten pixels both vertically and horizontally.

```
.s7zoomviewer .s7swatches .s7thumbcell {
  margin: 5px;
}
```

The appearance of the individual thumbnail is controlled with the following CSS class selector:

```
.s7zoomviewer .s7swatches .s7thumb
```

CSS property	Description
width	Width of the thumbnail.

CSS property	Description
height	Height of the thumbnail.
border	Border of the thumbnail.



Note: Thumbnail supports the `state` attribute selector, which can be used to apply different skins to different thumbnail states. In particular, `state="selected"` corresponds to the thumbnail for the image that is currently displayed in the main view, `state="default"` corresponds to the rest of thumbnails, and `state="over"` is used on mouse hover.

Example – to set up thumbnails that are 56 x 56 pixels, have a light grey default border, and a dark grey selected border.

```
.s7zoomviewer .s7swatches .s7thumb {
  width: 56px;
  height: 56px;
}
.s7zoomviewer .s7swatches .s7thumb[state="default"] {
  border: 1px solid #dddddd;
}
.s7zoomviewer .s7swatches .s7thumb[state="selected"] {
  border: 1px solid #666666;
}
```

The appearance of the left and right scroll buttons are controlled with the following CSS class selectors:

```
.s7zoomviewer .s7swatches .s7scrollleftbutton
.s7zoomviewer .s7swatches .s7scrollrightbutton
```

It is not possible to position scroll buttons using CSS top, left, bottom, and right properties. Instead, the viewer logic positions them automatically.

CSS property	Description
width	Width of the scroll button.
height	Height of the scroll button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states: up, down, over, and disabled.

The button tool tips can be localized. See [Localization of user interface elements](#).

Example – to set up scroll buttons that are 56 x 56 pixels and have different artwork for each state.

```
.s7zoomviewer .s7swatches .s7scrollleftbutton {
  background-size: auto;
```

```
width: 56px;
height: 56px;
}
.s7zoomviewer .s7swatches .s7scrollleftbutton[state="up"]{
background-image:url(images/v2/ScrollLeftButton_up.png);
}
.s7zoomviewer .s7swatches .s7scrollleftbutton[state="over"]{
background-image:url(images/v2/ScrollLeftButton_over.png);
}
.s7zoomviewer .s7swatches .s7scrollleftbutton[state="down"]{
background-image:url(images/v2/ScrollLeftButton_down.png);
}
.s7zoomviewer .s7swatches .s7scrollleftbutton[state="disabled"]{
background-image:url(images/v2/ScrollLeftButton_disabled.png);
}
.s7zoomviewer .s7swatches .s7scrollrightbutton {
background-size: auto;
width: 56px;
height: 56px;
}
.s7zoomviewer .s7swatches .s7scrollrightbutton[state="up"]{
background-image:url(images/v2/ScrollRightButton_up.png);
}
.s7zoomviewer .s7swatches .s7scrollrightbutton[state="over"]{
background-image:url(images/v2/ScrollRightButton_over.png);
}
.s7zoomviewer .s7swatches .s7scrollrightbutton[state="down"]{
background-image:url(images/v2/ScrollRightButton_down.png);
}
.s7zoomviewer .s7swatches .s7scrollrightbutton[state="disabled"]{
background-image:url(images/v2/ScrollRightButton_disabled.png);
}
```

Tooltips

On desktop systems some user interface elements like buttons have tooltips that are displayed on mouse hover.

CSS properties of the main viewer area

The appearance of tooltips is controlled with the following CSS class selector:

.s7tooltip

CSS property	Description
border-radius	Background border radius.
border-color	Background border color.
background-color	Background color.
color	Text color.
font-family	Text font name.
font-size	Text font size.



Note: In case tooltip styles are customized from within the embedding web page, all properties have to contain !IMPORTANT rule. This is not necessary if tooltips are customized in the viewer's CSS file.

Example – to set up tooltips that have a grey border with 3px corner radius, black background and white text written with Arial, 11 pixels size:

```
.s7tooltip {
border-radius: 3px 3px 3px 3px;
border-color: #999999;
background-color: #000000;
color: #FFFFFF;
font-family: Arial, Helvetica, sans-serif;
font-size: 11px;
}
```

Zoom in button

Clicking or tapping this button zooms in on an image in the main view. This button does not display on mobile phones in order to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7zoomviewer .s7zoominbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#).

Example – to set up a zoom in button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7zoomviewer .s7zoominbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7zoomviewer .s7zoominbutton [state='up'] {
background-image:url(images/v2/ZoomInButton_dark_up.png);
}
.s7zoomviewer .s7zoominbutton [state='over'] {
background-image:url(images/v2/ZoomInButton_dark_over.png);
}
.s7zoomviewer .s7zoominbutton [state='down'] {
background-image:url(images/v2/ZoomInButton_dark_down.png);
}
.s7zoomviewer .s7zoominbutton [state='disabled'] {
background-image:url(images/v2/ZoomInButton_dark_disabled.png);
}
```

Zoom out button

Clicking or tapping this button zooms out on an image in the main view. This button does not display on mobile phones in order to save screen real estate. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7zoomviewer .s7zoomoutbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#).

Example – to set up a zoom out button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7zoomviewer .s7zoomoutbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7zoomviewer .s7zoomoutbutton [state='up'] {
background-image:url(images/v2/ZoomOutButton_dark_up.png);
}
.s7zoomviewer .s7zoomoutbutton [state='over'] {
background-image:url(images/v2/ZoomOutButton_dark_over.png);
}
.s7zoomviewer .s7zoomoutbutton [state='down'] {
background-image:url(images/v2/ZoomOutButton_dark_down.png);
}
.s7zoomviewer .s7zoomoutbutton [state='disabled'] {
background-image:url(images/v2/ZoomOutButton_dark_disabled.png);
}
```

Zoom reset button


Clicking or tapping this button resets an image in the main view. You can size, skin, and position this button by using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7zoomviewer .s7zoomresetbutton
```

CSS property	Description
top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

 **Note:** This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#).

Example – to set up a zoom reset button that is 32 x 32 pixels, positioned six pixels from the top and right edge of the viewer, and displays a different image for each of the four different button states.

```
.s7zoomviewer .s7zoomresetbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7zoomviewer .s7zoomresetbutton [state='up'] {
background-image:url(images/v2/ZoomResetButton_dark_up.png);
}
.s7zoomviewer .s7zoomresetbutton [state='over'] {
background-image:url(images/v2/ZoomResettButton_dark_over.png);
}
.s7zoomviewer .s7zoomresetbutton [state='down'] {
background-image:url(images/v2/ZoomResetButton_dark_down.png);
}
.s7zoomviewer .s7zoomresetbutton [state='disabled'] {
background-image:url(images/v2/ZoomResetButton_dark_disabled.png);
}
```

Zoom view

Main view consists of the zoomable image.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7zoomviewer .s7zoomview
```

CSS property	Description
background-color	Background color in hexadecimal format of the main view.
cursor	Cursor displayed over the main view.

Example – to make the main view transparent.

```
.s7zoomviewer .s7zoomview {
background-color: transparent;
}
```

On desktop systems the component supports `cursor` attribute selector which can be applied to the `.s7zoomview` class. It controls the type of the cursor based on component state and user action. The following `cursor` values are supported:

- default
Displayed when the image is not zoomable because of a small image resolution, or component settings, or both.
- zoomin
Displayed when the image can be zoomed in.
- reset
Displayed when the image is at maximum zoom level and can be reset to its initial state.
- drag

Displayed when the user pans the image which is in zoomed in state.

- **slide**

Displayed when the user performs image swap by doing a horizontal swipe or flick.

Support for Adobe Analytics tracking

Out-of-the-box tracking

The Video Viewer supports Adobe Analytics tracking out-of-the-box. To enable tracking, pass the proper company preset name as `config2` parameter.

The viewer also sends a single tracking HTTP request to the configured Image Server with the viewer type and version information.

Custom tracking

To integrate with third-party analytics systems it is necessary to listen to the `trackEvent` viewer callback and process the `eventInfo` argument of the callback function as necessary. The following code is an example of such handler function:

```
var zoomViewer = new s7viewers.ZoomViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "Scene7SharedAssets/ImageSet-Views-Sample",
    "serverurl": "http://s7dl.scene7.com/is/image/"
  },
  "handlers": {
    "trackEvent": function(objID, compClass, instName, timeStamp, eventInfo) {
      //identify event type
      var eventType = eventInfo.split(",")[0];
      switch (eventType) {
        case "LOAD":
          //custom event processing code
          break;
        //additional cases for other events
      }
    }
  }
});
```

The viewer tracks the following SDK user events:

SDK user event	Sent when...
LOAD	viewer is loaded first.
SWAP	an asset is swapped in the viewer using the <code>setAsset()</code> API.
ZOOM	an image is zoomed.
PAN	an image is panned.
SWATCH	an image is changed by clicking or tapping on a swatch.

Localization of user interface elements

Certain content that the Video Viewer displays is subject to localization, including zoom buttons and a full screen button.

Every textual content in the viewer that can be localized is represented by a special Viewer SDK identifier called SYMBOL. Any SYMBOL has a default associated text value for the English locale ("en") supplied with the out-of-the-box viewer. It also may have user-defined values set for as many locales as needed.

When the viewer starts, it checks the current locale to see if there is a user-defined value for each supported SYMBOL for the locale. If there is, it uses the user-defined value; otherwise, it falls back to the out-of-the-box default text.

User-defined localization data can be passed to the viewer as a localization JSON object. Such object contains the list of supported locales, SYMBOL text values for each locale, and the default locale.

An example of such a localization object is the following:

```
{
  "en": {
    "CloseButton.TOOLTIP": "Close",
    "ZoomInButton.TOOLTIP": "Zoom In"
  },
  "fr": {
    "CloseButton.TOOLTIP": "Fermer",
    "ZoomInButton.TOOLTIP": "Agrandir"
  },
  defaultLocale: "en"
}
```

In the above example, the localization object defines two locales ("en" and "fr") and provides localization for two user interface elements in each locale.

The web page code should pass such localization object to the viewer constructor as a value of `localizedTexts` field of the configuration object. An alternative option is to pass the localization object by calling the `setLocalizedTexts(localizationInfo)` method.

The following SYMBOLs are supported:

SYMBOL	Tooltip for the...
CloseButton.TOOLTIP	Close button.
ZoomInButton.TOOLTIP	Zoom in button.
ZoomOutButton.TOOLTIP	Zoom out button.
ZoomResetButton.TOOLTIP	Zoom reset button.
FullScreenButton.TOOLTIP_SELECTED	Full screen button in normal state.
FullScreenButton.TOOLTIP_UNSELECTED	Full screen button in full screen state.
ScrollLeftButton.TOOLTIP	Scroll left button.

SYMBOL	Tooltip for the...
<code>ScrollRightButton.TOOLTIP</code>	Scroll right button.
<code>ScrollUpButton.TOOLTIP</code>	Scroll up button.
<code>ScrollDownButton.TOOLTIP</code>	Scroll down button.

Full Screen Support

The viewer supports full screen operation mode.

On modern desktop browsers, except Internet Explorer 10 and older, and on some touch devices, the viewer uses "native" full screen mode. This mode means that the entire device screen is occupied by the viewer content.

On iOS devices and on older Internet Explorer browsers, the viewer uses "simulated" full screen mode instead. In this mode, the viewer simply resizes to take the full area of the web browser window. Also, the web browser Chrome and other windows are still visible on the screen.

An end user enters and leaves full screen mode by pressing the Full Screen button in the viewer user interface. When "native" full screen mode is used on desktop, it is also possible to exit it by pressing **Esc**.

Viewer SDK namespace

The viewer is built of many Viewer SDK components. In most cases, the web page does not need to interact with SDK components API directly; all common needs are covered in the viewer API itself.

However, some advanced use cases require that the web page obtain a reference to an inner SDK component using the `getComponent()` viewer API and then use all the flexibility of the APIs of SDK itself.

The namespace that is used to load and initialize SDK components by the viewer depends on the environment in which the viewer is operating. If the viewer is running in AEM (Adobe Experience Manager), the viewer loads SDK components into `s7viewers.s7sdk` namespace. Likewise, the viewer served from Scene7 Publishing System loads the SDK into `s7classic.s7sdk`.

In either case, the namespace used by the SDK inside the viewer has either `s7viewers` or `s7classic` as the prefix. And, it is different from the plain `s7sdk` namespace used in the SDK User Guide or SDK API documentation. For that reason, it is important to use a fully qualified SDK namespace when you write custom application code that communicates with internal viewer components.

For example, if you plan to listen to `StatusEvent.NOTF_VIEW_READY` event and the viewer is served from AEM, the fully qualified event type is `s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY`, and the event listener code looks similar to the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var zoomView = <instance>.getComponent("zoomView");
    zoomView.addEventListener(s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});
```

The same code for viewer served from Scene7 SPS will look like this:

```
<instance>.setHandlers({
  "initComplete":function() {
    var zoomView = <instance>.getComponent("zoomView");
    zoomView.addEventListener(s7classic.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});
```

Viewers for AEM Assets only

Interactive Image

Interactive Image Viewer is a viewer that displays a single, non-zoomable image with clickable hotspots. The purpose of this viewer is to implement a "shoppable banner" experience. That is, the user can select a hotspot over the banner image and get redirected to a Quick View or product detail page on your website. It is designed to work on desktops and mobile devices.



Note: Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.

Viewer type is 508.

Demo URL

https://marketing.adobe.com/resources/help/en_US/dm/shoppable-banner/samples/InteractiveImage.html

System Requirements

See [System requirements](#).

Using Interactive Image Viewer

Interactive Image Viewer represents a main JavaScript file and a set of helper files (a single JavaScript include with all Viewer SDK components used by this particular viewer, assets, CSS) downloaded by the viewer in runtime.

Interactive Image Viewer can be used in embedded mode only, where it is integrated into the target web page using the documented API.

Configuration and skinning are similar to that of the other viewers described in this Help. All skinning is achieved by way of custom CSS.

See [Command reference common to all viewers – Configuration attributes](#) and [Command reference common to all Viewers – URL](#)

Interacting with Interactive Image Viewer

Interaction that is supported by the Video Image Viewer is hotspot activation on desktop systems. This activation occurs on click and on touch devices with a single tap.

The viewer is fully keyboard accessible.

See [Keyboard accessibility and navigation](#).

Embedding Interactive Image Viewer

Interactive Image Viewer is designed to be embedded into the hosting page. Such a web page may have a static layout, or it may be "responsive" and display differently on different devices or for different browser window sizes.

To accommodate these needs, the viewer supports two primary operation modes: fixed size embedding and responsive embedding.

About fixed size embedding mode and responsive design embedding mode

In the embedded mode, the viewer is added to the existing web page. This web page may already have some customer content not related to the viewer. The viewer normally occupies only a part of a web page's real estate.

The primary use cases are web pages oriented for desktops or tablet devices, and responsive designed pages that adjust layout automatically depending on the device type.

Fixed size embedding is used when the viewer does not change its size after initial load. This is the best choice for web pages that have a static layout.

Responsive design embedding assumes that the viewer may need to resize at runtime in response to the size change of its container `DIV`. The most common use case is adding a viewer to a web page that uses a flexible page layout.

In responsive design embedding mode, the viewer behaves differently depending on the way web page sizes its container `DIV`. If the web page sets only the width of the container `DIV`, leaving its height unrestricted, the viewer automatically chooses its height according to the aspect ratio of the asset used. This functionality ensures that the asset fits perfectly into the view without any padding on the sides. This use case is the most common for web pages using responsive web design layout frameworks like Bootstrap, Foundation, and so on.

Otherwise, if the web page sets both the width and the height for the viewer's container `DIV`, the viewer fills just that area. It also follows the size that the web page layout provides. A good example is embedding the viewer into a modal overlay, where the overlay is sized according to web browser window size.

Fixed size embedding

You add the viewer to a web page by doing the following:

1. Adding the viewer JavaScript file to your web page.
2. Defining the container `DIV`.
3. Setting the viewer size.
4. Creating and initializing the viewer.

1. Adding the viewer JavaScript file to your web page.

Creating a viewer requires that you add a script tag in the HTML head. Before you can use the viewer API, be sure that you include `InteractiveImage.js`. The `InteractiveImage.js` file is located under the `html5/js/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/etc/dam/viewers/s7viewers/html5/js/InteractiveImage.js
```

You can use a relative path if the viewer is deployed on one of the Adobe Scene7 servers and it is served from the same domain. Otherwise, you specify a full path to one of Adobe Scene7 servers that have the IS-Viewers installed.

The relative path looks like the following:

```
<script language="javascript" type="text/javascript"
src="/etc/dam/viewers/s7viewers/html5/js/InteractiveImage.js"></script>
```



Note: You should only reference the main viewer JavaScript `include` file on your page. You should not reference any additional JavaScript files in the web page code which might be downloaded by the viewer's logic in runtime. In particular, do not directly reference `HTML5 SDK Utils.js` library loaded by the viewer from `/s7viewers` context path (so-called consolidated SDK `include`). The reason is that the location of `Utils.js` or similar runtime viewer libraries is fully managed by the viewer's logic and the location changes between viewer releases. Adobe does not keep older versions of secondary viewer `includes` on the server.

As a result, putting a direct reference to any secondary JavaScript `include` used by the viewer on the page breaks the viewer functionality in the future when a new product version is deployed.

2. Defining the container DIV.

Add an empty DIV element to the page where you want the viewer to appear. The DIV element must have its ID defined because this ID is passed later to the viewer API. The DIV has its size specified through CSS.

The placeholder DIV is a positioned element, meaning that the `position` CSS property is set to `relative` or `absolute`.

The following is an example of a defined placeholder DIV element:

```
<div id="s7viewer" style="position:relative"></div>
```

3. Setting the viewer size

You can set the static size for the viewer by either declaring it for `.s7interactiveimage` top-level CSS class in absolute units, or by using `stagesize` modifier.

You can put sizing in CSS directly on the HTML page, or in a custom viewer CSS file, which is then later assigned to a viewer preset record in AEM Assets – on-demand, or passed explicitly using `style` command.

See [Video](#) for more information about styling the viewer with CSS.

The following is an example of defining a static viewer size in the HTML page:

```
#s7viewer.s7interactiveimage {
  width: 1174px;
  height: 500px;
}
```

You can pass explicitly the `stagesize` modifier with the viewer initialization code with `params` collection or as an API call as described in the Command Reference section, like this:

```
interactiveImage.setParam("stagesize", "1174,500");
```

A CSS-based approach is recommended and is used in this example.

4. Creating and initializing the viewer.

When you have completed the steps above, you create an instance of `s7viewers.InteractiveImage` class, pass all configuration information to its constructor, and call `init()` method on a viewer instance. Configuration information is passed to the constructor as a JSON object. At minimum, this object should have `containerId` field which holds the name of viewer container ID and nested `params` JSON object with configuration parameters supported by the viewer. In this case, the `params` object must have at least the Image Serving URL passed as `serverUrl` property, and the initial asset as `asset` parameter. The JSON-based initialization API lets you create and start the viewer with a single line of code.

It is important to have the viewer container added to the DOM so that the viewer code can find the container element by its ID. Some browsers delay building DOM until the end of the web page. For maximum compatibility, call the `init()` method just before the closing BODY tag, or on the `body onload()` event.

At the same time, the container element should not necessarily be part of the web page layout just yet. For example, it may be hidden using `display:none` style assigned to it. In this case, the viewer delays its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load automatically resumes.

The following is an example of creating a viewer instance, passing minimum necessary configuration options to the constructor and calling the `init()` method. The example assumes `interactiveImage` is the viewer instance; `s7viewer` is the name of placeholder DIV; `http://aodmarketingna.assetsadobe.com/is/image` is the Image Serving URL, and `/content/dam/mac/aodmarketingna/shoppable-banner/shoppable-banner` is the asset:

```
<script type="text/javascript">
var interactiveImage = new s7viewers.InteractiveImage ({
```

```
"containerId": "s7viewer",
"params": {
  "asset": "/content/dam/mac/aodmarketingna/shoppable-banner/shoppable-banner.jpg",
  "serverurl": "http://aodmarketingna.assetsadobe.com/is/image"
}
}).init();
</script>
```

The following code is a complete example of a trivial web page that embeds the Video Image Viewer with a fixed size:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://aodmarketingna.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/InteractiveImage.js"></script>
<style type="text/css">
#s7viewer.s7interactiveimage {
  width: 1174px;
  height: 500px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position: relative"></div>
<script type="text/javascript">
var interactiveImage = new s7viewers.InteractiveImage({
  "containerId": "s7viewer",
  "params": {
    "asset": "/content/dam/mac/aodmarketingna/shoppable-banner/shoppable-banner.jpg",
    "serverurl": "http://aodmarketingna.assetsadobe.com/is/image"
  }
}).init();
</script>
</body>
</html>
```

Responsive design embedding with unrestricted height

With responsive design embedding, the web page normally has some kind of flexible layout in place that dictates the runtime size of the viewer's container DIV. For the following example, assume that the web page allows the viewer's container DIV to take 40% of the web browser window size. And, its height is left unrestricted. The web page HTML code would look like the following:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
.holder {
  width: 40%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>
```

Adding the viewer to such a page is similar to the steps for fixed size embedding. The only difference is that you do not need to explicitly define the viewer size.

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Creating and initializing the viewer.

All the steps above are the same as with the fixed size embedding. Add the container `DIV` to the existing "holder" `DIV`. The following code is a complete example. Notice how the viewer size changes when the browser is resized, and how the viewer aspect ratio matches the asset.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://aodmarketingna.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/InteractiveImage.js"></script>
<style type="text/css">
.holder {
width: 40%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var interactiveImage = new s7viewers.InteractiveImage({
  "containerId":"s7viewer",
  "params":{
    "asset":"/content/dam/mac/aodmarketingna/shoppable-banner/shoppable-banner.jpg",
    "serverurl":"http://aodmarketingna.assetsadobe.com/is/image"
  }
}).init();
</script>
</body>
</html>
```

The following examples page illustrates more real-life uses of responsive design embedding with unrestricted height:

https://marketing.adobe.com/resources/help/en_US/dm/shoppable-banner/samples/InteractiveImage-responsive-unrestricted-height.html

Flexible size Embedding with Width and Height Defined

In case of flexible-size embedding with width and height defined, the web page styling is different. It provides both sizes to the "holder" `DIV` and center it in the browser window. Also, the web page sets the size of the `HTML` and `BODY` element to 100 percent.

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
width: 100%;
height: 100%;
}
.holder {
position: absolute;
left: 20%;
top: 20%;
width: 60%;
height: 60%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>
```

The rest of the embedding steps are identical to the steps used for responsive embedding with unrestricted height. The resulting example is the following:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://aodmarketingna.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/InteractiveImage.js"></script>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
height: 60%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var interactiveImage = new s7viewers.InteractiveImage({
  "containerId":"s7viewer",
"params":{
  "asset":"/content/dam/mac/aodmarketingna/shoppable-banner/shoppable-banner.jpg",
  "serverurl":"http://aodmarketingna.assetsadobe.com/is/image"
}
}).init();
</script>
</body>
</html>
```

Embedding Using Setter-based API

Instead of using JSON-based initialization, it is possible to use setter-based API and no-args constructor. Using this API constructor does not take any parameters and configuration parameters are specified using `setContainerId()`, `setParam()`, and `setAsset()` API methods with separate JavaScript calls.

The following example illustrates using fixed size embedding with the setter-based API:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://aodmarketingna.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/InteractiveImage.js"></script>
<style type="text/css">
#s7viewer.s7interactiveimage {
  width: 1174px;
  height: 500px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var interactiveImage = new s7viewers.InteractiveImage();
interactiveImage.setContainerId("s7viewer");
interactiveImage.setParam("serverurl", "http://aodmarketingna.assetsadobe.com/is/image");
interactiveImage.setAsset("/content/dam/mac/aodmarketingna/shoppable-banner/shoppable-banner.jpg");
interactiveImage.init();
</script>
```



```
</body>
</html>
```

Command reference – Configuration attributes

Configuration attributes documentation for Interactive Image Viewer.

Any configuration command can be set in URL or using `setParam()`, or `setParams()`, or both, API methods. Any config attribute can be also specified in the server-side configuration record.

Some configuration commands may be prefixed with the class name or instance name of corresponding Viewer SDK component. An instance name of the component is dynamic and depends on the ID of the viewer container DOM element passed to `setContainerId()` API method. Documentation includes an optional prefix for such commands. For example, `zoomstep` command is documented as follows:

```
[ZoomView.|<containerId>_zoomView].fmt
```

which means that you can use this command as:

- `fmt` (short syntax)
- `ZoomView.fmt` (qualified with component class name)
- `cont_zoomView.fmt` (qualified with component ID, assuming `cont` is the ID of the container element)

See also [Command reference common to all viewers – Configuration attributes](#)

ZoomView.enableHD

```
[ZoomView.|<containerId>_zoomView.]enableHD=always|never|limit[,number]
```

<code>always never limit</code>	Enable, limit, or disable optimization for devices where <code>devicePixelRatio</code> is greater than 1. Affects devices with high-density display like iPhone4 and similar devices. If active, the component limits the size of the IS image request as if the device had a pixel ratio of 1, thereby reducing the bandwidth.
<code>number</code>	If using the limit setting, the component enables high pixel density only up to the specified limit.

Properties

Optional.

Default

```
limit,1500
```

Example

```
enableHD=always
```

ZoomView.fmt

Specifies the image format that the component uses for loading images from Image Server.

```
[ZoomView.<containerId>_zoomView.<fmt>=jpg|jpeg|png|png-alpha|gif|gif-alpha
```

<code>jpg jpeg png png-alpha gif gif-alpha</code>	If the specified format ends with <code>-alpha</code> , the component renders images as transparent content. For all other image formats the component treats images as opaque. The component has a white background by default. Therefore, to make it transparent, set the <code>background-color</code> CSS property to transparent.
---	--

Properties

Optional.

Default

jpeg

Example

```
fmt=png-alpha
```

ZoomView.iscommand

The Image Serving command string that is applied to zoom image.

```
[ZoomView.<containerId>_zoomView.<iscommand>=isCommand
```

<code>iscommand</code>	If specified in the URL all occurrences of <code>&</code> and <code>=</code> must be HTTP-encoded as <code>%26</code> and <code>%3D</code> , respectively.
------------------------	--

Properties

Optional.

Default

None.

Example

When specified in the viewer URL:

```
iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000
```

When specified in the config data:

```
iscommand=op_sharpen=1&op_colorize=0xff0000
```

Command reference – URL

Command reference documentation for Video Video Viewer.

preloadimage

URL command for Video Image Viewer.

preloadImage=0 | 1

0 1	Enable (1) or disable (0) the preload image feature. See Preload image .
-------	---

Properties

Optional.

Default

1

Example

```
preloadImage=0
```

JavaScript API reference for Interactive Image Viewer

The main class of the Interactive Image Viewer is `InteractiveImage`. It is declared in the `s7viewers` namespace. This JavaScript API covers constructor, methods, and callbacks of this particular class.

In all the following examples, `<instance>` stands for the actual name of the JavaScript viewer object that is instantiated from the `s7viewers.InteractiveImage` class.

dispose

JavaScript API reference for Video Image Viewer.

```
dispose()
```

Disposes this viewer instance by releasing all resources used by the viewer logic and deleting all inner objects and components created by the viewer in runtime.

The web page code should also delete the viewer instance variable as well to completely remove the viewer from the web browser memory.

If the web page code has registered event listeners directly on Viewer SDK components used by the viewer—or stored external references to such components—such listeners must be explicitly unregistered by the web page code, and such external component references must be deleted prior to calling `dispose()`.

Do not access the Viewer API any more after `dispose()` is called.

Parameters

None.

Returns

None.

Example

```
<instance>.dispose()
```

getComponent

JavaScript API reference for Video Image Viewer.

```
getComponent(componentId)
```

Returns a reference to the Viewer SDK component that is used by the viewer. The web page can use this method to extend or customize the behavior of the out-of-box viewer. Call this method only after the `initComplete` viewer callback has run, otherwise the component may not be created yet by the viewer logic.

Parameters

componentID - {String} an ID of the Viewer SDK component used by the viewer. This viewer supports the following component IDs:

Component ID	Viewer SDK component class name
parameterManager	s7sdk.ParameterManager
container	s7sdk.common.Container
mediaSet	s7sdk.set.MediaSet
zoomView	s7sdk.image.ZoomView
imageMapEffect	s7sdk.image.mageMapEffect

When working with SDK APIs it is important to use the correct fully qualified SDK namespace as described in [Viewer SDK namespace](#).

Refer to the Viewer SDK API documentation for more information about a particular component.

Returns

{Object} a reference to Viewer SDK component. The method returns `null` if the `componentId` is not a supported viewer component or if the component was not yet created by the viewer logic.

Example

```
<instance>.setHandlers({  
  "initComplete":function() {  
    var zoomView = <instance>.getComponent("zoomView");  
  }  
})
```

init

JavaScript API reference for Video Image Viewer.

```
init()
```

Starts the initialization of the Video Image Viewer. By this time the container DOM element must be created so that the viewer code can find it by its ID.

If the container element is not a part of the web page layout just yet (for example, it may be hidden using `display:none` style assigned to it), the viewer suspends its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load automatically resumes.

Call this method only once during viewer life cycle; subsequent calls are ignored.

Parameters

None.

Returns

{Object} A reference to the viewer instance.

Example

```
<instance>.init()
```

InteractiveImage

JavaScript API reference for Video Image Viewer.

```
InteractiveImage([config])
```

Constructor, creates a new Video Image Viewer instance.

Parameters

<i>config</i>	<p>{object} optional JSON configuration object, allows all the viewer settings to pass to the constructor to avoid calling individual setter methods. Contains the following properties:</p> <ul style="list-style-type: none">• containerId – {String} ID of the DOM container (normally a <code>DIV</code>) that the viewer is inserted into. By the time this method is called, it is not necessary to have the container element created. However, the container must exist when <code>init()</code> is run. Required.• params – {Object} JSON object with viewer configuration parameters where the property name is either viewer-specific configuration option or SDK modifier, and the value of that property is a corresponding settings value. Required.• handlers – {Object} JSON object with viewer event callbacks, where the property name is the name of supported viewer event and the property value is a JavaScript function reference to appropriate callback. Optional. See Event callbacks for more information about viewer events.
---------------	--

Returns

None.

Example

```
var interactiveImage = new s7viewers.InteractiveImage({
  "containerId": "s7viewer",
  "params": {
    "asset": "/content/dam/mac/aodmarketingna/shoppable-banner/shoppable-banner.jpg",
    "serverurl": "http://aodmarketingna.assetsadobe.com/is/image"
  },
  "handlers": {
    "initComplete": function() {
      console.log("init complete");
    }
  }
});
```

setAsset

JavaScript API reference for Video Image Viewer.

```
setAsset(asset)
```

Sets the new asset. You can call this parameter at any time, either before or after `init()`. If it is called after `init()`, the viewer swaps the asset at runtime.

See also [init](#).

<code>asset</code>	<code>{String}</code> new asset ID. Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.
--------------------	--

Returns

None.

Example

Single image reference:

```
<instance>.setAsset("/content/dam/mac/aodmarketingna/shoppable-banner/shoppable-banner.jpg")
```

setContainerId

JavaScript API reference for Video Image Viewer.

```
setContainerId(containerId)
```

Sets the ID of the DOM container (normally a DIV) into which the viewer is inserted. It is not necessary to have the container element created by the time this method is called. However, the container must exist when `init()` is run. It must be called before `init()`.

This method is optional if the viewer configuration information is passed with the `config` JSON object to the constructor.

Parameter

<code>containerId</code>	<code>{string}</code> ID of container.
--------------------------	--

Returns

None.

Example

```
<instance>.setContainerId("s7viewer");
```

setHandlers

JavaScript API reference for Video Image Viewer

```
setHandlers(handlers)
```

Specifies zero or more callback handlers. A call to this method fully overwrites event handlers that were previously assigned for that viewer instance. Must be called before `init()`.

Parameter

<i>handlers</i>	<p>{Object} JSON object with viewer event callbacks. The property name is the name of the supported viewer event. The property value is a JavaScript function reference to an appropriate callback.</p> <p>See Event callbacks for more information about viewer events.</p>
-----------------	--

Returns

None.

Example

```
<instance>.setHandlers({  
  "initComplete":function() {  
    console.log("init complete");  
  }  
})
```

setParam

JavaScript API reference for Video Image Viewer.

```
setParam(name, value)
```

Sets the viewer parameter to a specified value. The parameter is either a viewer-specific configuration option or a software development kit modifier. This parameter is called before `init()`.

This method is optional if the viewer configuration information was passed with `config` JSON object to constructor.

See also [init](#).

Parameters

<i>name</i>	{string} name of parameter.
<i>value</i>	{string} value of parameter. The value cannot be percent-encoded.

Returns

None.

Example

```
<instance>.setParam("style", "etc/dam/presets/css/html5_interactiveimage.css")
```

setParams

JavaScript API reference for Video Image Viewer.

```
setParams(params)
```

Sets one or more parameters to a given value. The method argument syntax is identical to a URL query string. That is, it represents name=value pairs separated with &. Just as in a query string, the names and values are percent-encoded using UTF8. Before you call `init()`, this parameter must be called.

This method is optional if the viewer configuration information was passed with `config` JSON object to constructor.

See also [init](#).

Parameter

<i>params</i>	{string} name=value parameter pairs separated with &.
---------------	---

Returns

None.

Example

```
<instance>.setParams("ZoomView.fmt=png&ZoomView.iscommand=op_sharpen%3d1")
```

Event callbacks

The viewer supports JavaScript event callbacks that the web page uses to track the viewer initialization process or runtime behavior.

Callback handlers are assigned by passing event names and corresponding handler functions with the `handlers` property to `config` JSON object in the viewer's constructor. Alternatively, it is possible to use `setHandlers()` API method.

Supported viewer events include the following:

Viewer event	Description
initComplete	Triggers when viewer initialization is complete and all internal components are created, so that it is possible to use <code>getComponent()</code> API. The callback handler does not take any arguments.
trackEvent	Triggers each time an event occurs inside the viewer which may be handled by an event tracking system, such as Adobe Analytics. The callback handler accepts the following arguments: <ul style="list-style-type: none">objID {String} – not currently used.

Viewer event	Description
	<ul style="list-style-type: none"> • <code>compClass {String}</code> – not currently used. • <code>instName {String}</code> – an instance name of the Viewer SDK component that triggered the event. • <code>timeStamp {Number}</code> – event time stamp. • <code>eventInfo {String}</code> – event payload.
<code>quickViewActivate</code>	<p>Triggers when the user activates a hotspot with Quick View data associated with it. The callback handler takes the following argument:</p> <ul style="list-style-type: none"> • <code>data {Object}</code> – a JSON object containing data from the hotspot definition. The field <code>sku</code> is mandatory while other fields are optional and depend on the source hotspot definition.

See also [InteractiveImage](#) and [setHandlers](#).

Customizing Interactive Image Viewer

All visual customization and most behavior customization for the Interactive Image Viewer is done by creating a custom CSS.

The suggested workflow is to take the default CSS file for the appropriate viewer, copy it to a different location, customize it, and specify the location of the customized file in the `style=` command.

Default CSS files can be found at the following location:

```
<s7viewers_root>/etc/dam/viewers/s7viewers/html5/InteractiveImage.css
```

Custom CSS file must contain the same class declarations as the default one. If a class declaration is omitted, the viewer does not function properly because it does not provide built-in default styles for user interface elements.

Alternative way to provide custom CSS rules is to use embedded styles directly on the web page or in one of linked external CSS rules.

When creating custom CSS keep in mind that the viewer assigns `.s7interactiveimage` class to its container DOM element. If you are using an external CSS file passed with the `style=` command, use `.s7interactiveimage` class as parent class in descendant selector for your CSS rules. If you are adding embedded styles on the web page, additionally qualify this selector with an ID of the container DOM element as follows:

```
#<containerId>.s7interactiveimage
```

Building responsive designed CSS

It is possible to target different devices and embedding sizes in CSS to make your content display differently, depending on a user's device or a particular web page layout. This includes, but is not limited to, different layouts, user interface element sizes, and artwork resolution.

The viewer supports two mechanisms of creating responsive designed CSS: CSS markers and standard CSS media queries. You can use these independently or together.

CSS markers

To assist in creating responsive designed CSS, the viewer supports CSS markers. These are special CSS classes that are dynamically assigned to the top-level viewer container element. They are based on the runtime viewer size and the input type used on the current device.

The first group of CSS markers contains `.s7size_large`, `.s7size_medium`, and `.s7size_small` classes. They are applied based on the runtime area of the viewer container. For example, if the viewer area is equal or bigger than the size of a common desktop monitor, use `.s7size_large`. If the area is close to a common tablet device, assign `.s7size_medium`. For areas similar to mobile phone screens, use `.s7size_small`. The primary purpose of these CSS markers is to create different user interface layouts for different screens and viewer sizes.

The second group of CSS markers contains `.s7mouseinput` and `.s7touchinput`. The CSS marker `.s7touchinput` is set if the current device is capable of touch input. Otherwise, `.s7mouseinput` is used. These markers are mostly intended to create user interface input elements with different screen sizes for different input types, because normally touch input requires larger elements.

The following sample CSS sets the zoom in button size to 28 x 28 pixels on systems with mouse input and to 56 x 56 pixels on touch input devices. If the viewer is sized even smaller, it is set to 20 x 20 pixels.

```
.s7interactiveimage.s7mouseinput .s7imagemapeffect .s7icon {
    width:28px;
    height:28px;
}
.s7interactiveimage.s7touchinput .s7imagemapeffect .s7icon {
    width:56px;
    height:56px;
}
.s7interactiveimage.s7size_small .s7imagemapeffect .s7icon {
    width:20px;
    height:20px;
}
```

To target devices with different pixel density you need to use CSS media queries. The following media query block would contain CSS specific to high-density screens:

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5)
{
}
```

Using CSS markers is the most flexible way of building responsive designed CSS because it lets you target not only the device screen size but the actual viewer size, which is useful for responsive design layouts.

You can use the default viewer CSS file as an example of a CSS markers approach.

CSS media queries

You can also accomplish device sensing by using pure CSS media queries. Everything enclosed within a given media query block is applied only when it is run on a corresponding device.

When applied to Mobile Viewers use four CSS media queries, defined in your CSS, in the order listed below:

1. Contains only rules specific for all touch devices.

```
@media only screen and (max-device-width:13.5in) and (max-device-
height:13.5in) and (max-device-width:799px),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in)
and (max-device-height:799px)
{
}
```

2. Contains only rules specific for tablets with high-resolution screens.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px) and (-webkit-min-device-pixel-ratio:1.5),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px) and (-webkit-min-device-pixel-ratio:1.5)
{
}
```

3. Contains only rules specific for all mobile phones.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in)
{
}
```

4. Contains only rules specific for mobile phones with high-resolution screens.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in) and
(-webkit-min-device-pixel-ratio: 1.5),
only screen and (device-width:720px) and (device-height:1280px) and
(-webkit-device-pixel-ratio: 2),
only screen and (device-width:1280px) and (device-height:720px) and
(-webkit-device-pixel-ratio: 2)
{
}
```

Using a media queries approach, you should organize CSS with device sensing as follows:

- First, the desktop-specific section defines all properties that are either desktop-specific or common to all screens.
- And second, the four media queries go in the order defined above and provide CSS rules that are specific for the corresponding device type.

There is no need to duplicate the entire viewer CSS in each media query. Only properties that are specific to given devices are redefined inside a media query.

CSS sprites

Many viewer user interface elements are styled using bitmap artwork and have more than one distinct visual state. A good example is a button that normally has at least three different states: up, over, and down. Each state requires its own bitmap artwork assigned.

With a classic approach to styling, the CSS would have a separate reference to the individual image file on the server for each state of the user interface element. The following is a sample CSS for styling a zoom-in button:

```
.s7interactiveimage .s7imagemapeffect .s7icon {
background-image: url(images/v2/imagemap/ImageMapEffect_icon1_light_up_touch.png);
}
.s7interactiveimage .s7imagemapeffect .s7icon:hover {
background-image: url(images/v2/imagemap/ImageMapEffect_icon1_light_over_touch.png);
}
```

The drawback to this approach is that the end user experiences flickering or delayed user interface response when the element is interacted with for the first time. This action occurs because the image artwork for the new element state is not yet downloaded. Also, this approach may have a slight negative impact on performance because of an increase in the number of HTTP calls to the server.

CSS sprites is a different approach where image artwork for all element states is combined into a single PNG file called a "sprite". Such "sprite" has all visual states for the given element positioned one after another. When styling a user interface element with sprites the same sprite image is referenced for all different states in the CSS. Also, the `background-position` property is used

for each state to specify which part of the "sprite" image is used. You can structure a "sprite" image in any suitable way. Viewers normally have it vertically stacked.

The following is a "sprite"-based example of styling the same zoom-in button earlier:

```
.s7interactiveimage .s7imagemapeffect .s7icon {
background-image: url(images/v2/imagemap/ImageMapEffect_icon1_light_up_touch.png);
background-position: -0px -56px; width: 56px; height: 56px;
}
.s7interactiveimage .s7imagemapeffect .s7icon: hover {
background-position: -0px -0px; width: 56px; height: 56px;
}
```

General styling notes and advice

- When customizing the viewer user interface with CSS the use of the `!IMPORTANT` rule is not supported to style viewer elements. In particular, `!IMPORTANT` rule should not be used to override any default or runtime styling provided by the viewer or Viewer SDK. The reason for this is that it may affect the behavior of proper components. Instead, you should use CSS selectors with the proper specificity to set CSS properties that are documented in this Viewers Reference guide.
- All paths to external assets within CSS are resolved against the CSS location, not the viewer HTML page location. Be aware of this rule when you copy the default CSS to a different location. Either copy the default assets as well or update all the paths within the custom CSS.
- The preferred format for bitmap artwork is PNG.
- Bitmap artwork is assigned to user interface elements using the `background-image` property.
- The `width` and `height` properties of a user interface element define its logical size. The size of the bitmap passed to `background-image` does not affect its logical size.
- To use the high pixel density of high-resolution screens like Retina, specify bitmap artwork twice as large as the logical user interface element size. Then, apply the `-webkit-background-size: contain` property to scale the background down to the logical user interface element size.
- To remove a button from the user interface, add `display: none` to its CSS class.
- You can use various formats for color value that CSS supports. If you need transparency, use the format `rgba(R, G, B, A)`. Otherwise, you can use the format `#RRGGBB`.

Common user interface elements

The following is user interface elements reference documentation that applies to the Video Image Viewer:

Focus highlight

Input focus highlight displayed around focused viewer UI element is controlled with the CSS class selector.

CSS properties

The appearance is controlled with the following CSS class selector:

```
.s7interactiveimage *:focus
```

CSS property	Description
outline	Focus highlight style.

Example – to disable the default browser focus highlight for all viewer user interface elements, add the following CSS selector to viewer's style sheet:

```
.s7interactiveimage *:focus {
  outline: none;
}
```

Hotspots

The viewer displays hotspot icons over the main view in places where hotspots were originally authored in Dynamic Media of AEM Assets – on-demand.

CSS properties of the main viewer area

The appearance of the hotspot icon is controlled with the following CSS class selector:

```
.s7interactiveimage .s7imagemapeffect .s7icon
```

CSS property	Description
background-image	Hotspot icon artwork.
background-position	Position inside the artwork sprite, if CSS sprites are use. See CSS sprites .
width	Hotspot icon width.
height	Hotspot icon height.

Example – set up a 56 x 56 pixels hotspot icon that displays a different image for each of the two different icon states:

```
.s7interactiveimage .s7imagemapeffect .s7icon {
  background-image: url(images/v2/imagemap/ImageMapEffect_icon1_light_up_touch.png);
  width: 56px;
  height: 56px;
}
.s7interactiveimage .s7imagemapeffect .s7icon: hover {
  background-image: url(images/v2/imagemap/ImageMapEffect_icon1_light_over_touch.png);
}
```

Main viewer area

The main view area is the area occupied by the zoom image. It is usually set to fit the available device screen when no size is specified.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7interactiveimage
```

CSS property	Description
width	The width of the viewer.

CSS property	Description
height	The height of the viewer.
background-color	Background color in hexadecimal format.

Example – to set up a viewer with a white background (#FFFFFF) and make its size 1174 x 500 pixels.

```
.s7interactiveimage {
  background-color: #FFFFFF;
  width: 1174px;
  height: 500px;
}
```

Tooltips

On desktop systems some user interface elements like buttons have tool tips that are displayed on mouse hover.

CSS properties of the main viewer area

The appearance of tool tips is controlled with the following CSS class selector:

```
.s7tooltip
```

CSS property	Description
border-radius	Background border radius.
border-color	Background border color.
background-color	Background color.
color	Text color.
font-family	Text font name.
font-size	Text font size.



Note: In case tool tip styles are customized from within the embedding web page, all properties have to contain !IMPORTANT rule. This is not necessary if tool tips are customized in the viewer's CSS file.

Example – to set up tool tips that have a gray border with a 3 pixel corner radius, black background, and white text in Arial, 11 pixels size:

```
.s7tooltip {
  border-radius: 3px 3px 3px 3px;
  border-color: #999999;
  background-color: #000000;
  color: #FFFFFF;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 11px;
}
```

Zoom view

Main view consists of the static image.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7interactiveimage .s7zoomview
```

CSS property	Description
background-color	Background color in hexadecimal format of the main view.

Example – to make the main view transparent.

```
.s7interactiveimage .s7zoomview {
  background-color: transparent;
}
```

Support for analytics tracking

Custom tracking

By default, the viewer sends a single tracking HTTP request to configured Image Server with the viewer type and version information.

To integrate with third-party analytics systems, it is necessary to listen to the `trackEvent` viewer callback and process the `eventInfo` argument of the callback function as necessary. The following code is an example of such handler function:

```
var interactiveImage = new s7viewers.InteractiveImage({
  "containerId": "s7viewer",
  "params": {
    "asset": "/content/dam/mac/aodmarketingna/shoppable-banner/shoppable-banner.jpg",
    "serverurl": "http://aodmarketingna.assetsadobe.com/is/image"
  },
  "handlers": {
    "trackEvent": function(objID, compClass, instName, timeStamp, eventInfo) {
      //identify event type
      var eventType = eventInfo.split(",")[0];
      switch (eventType) {
        case "LOAD":
          //custom event processing code
          break;
        //additional cases for other events
      }
    }
  }
});
```

The viewer tracks the following SDK user events:

SDK user event	Sent when...
LOAD	viewer is loaded first.
HREF	user activates hotspot.

Hotspot support

The viewer supports the rendering of hotspot icons on top of the main view. The appearance of hotspot icons is controlled through CSS as described in the Hotspots section.

See [Hotspots](#).

Hotspots can either activate a Quick View feature on the hosting web page by triggering a JavaScript callback or redirect a user to an external web page.

Quick View hotspots

These types of hotspots should be authored using the "Quick View" action type in Dynamic Media, of AEM Assets – on demand. When a user activates such a hotspot, the viewer runs the `quickViewActivate` JavaScript callback and passes the hotspot data to it. It is expected that the embedding web page listens for this callback. When it triggers the page, it opens its own Quick View implementation.

Redirect to external web page

Hotspots authored for the action type "Quick View" in Dynamic Media of AEM Assets – on demand redirects the user to an external URL. Depending on settings made during authoring, the URL opens in a new browser tab, in the same window, or in the named browser window.

Preload image

Preload image is a static asset preview image which loads right after calling `init()` method and shows while Viewer SDK libraries, asset and preset information is downloaded. The purpose of the preload image is to visually improve viewer load time and present content to the user quickly.

Preload image works well for the most common viewer embedding method, which is responsive embedding with unrestricted height. See the heading [Responsive design embedding with unrestricted height](#).

The feature, however, has certain limitations when other embedding methods or specific configuration options are used. Preload image may fail to render properly in following cases:

- When the viewer is fixed in size and the size is defined either using `stagesize` configuration attribute inside the viewer preset record or in the external viewer CSS file for the top-level viewer container element.
- When the flexible size embedding with width and height defined method of viewer embedding is used. See the heading [Flexible size embedding with width and height defined](#).

You may need to disable preload image feature using the `preloadImage` configuration attribute if you are using the viewer in one of the operation modes listed above.

Also, preload image is not used—even if enabled in the configuration—if the viewer is embedded into the DOM element is hidden using `display:none` CSS setting or detached from the DOM tree.

Viewer SDK namespace

The viewer is built of many Viewer SDK components. In most cases, the web page does not need to interact with SDK components API directly; all common needs are covered in the viewer API itself.

However, some advanced use cases require that the web page obtain a reference to an inner SDK component using the `getComponent()` viewer API and then use all the flexibility of the APIs of SDK itself.

The namespace that is used to load and initialize SDK components by the viewer depends on the environment in which the viewer is operating. If the viewer is running in AEM (Adobe Experience Manager), the viewer loads SDK components into `s7viewers.s7sdk` namespace. And the viewer served from Scene7 Publishing System loads the SDK into `s7classic.s7sdk`.

In either case, the namespace used by the SDK inside the viewer has either `s7viewers` or `s7classic` as the prefix. And, it is different from the plain `s7sdk` namespace used in the SDK User Guide or SDK API documentation.

For that reason it is important to use a fully qualified SDK namespace when you write custom application code that communicates with internal viewer components.

For example, if you plan to listen to `StatusEvent.NOTF_VIEW_READY` event and the viewer is served from AEM, the fully qualified event type is `s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY`, and the event listener code looks similar to the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var zoomView = <instance>.getComponent("zoomView");
    zoomView.addEventListener(s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e) {
      console.log("view ready");
    }, false);
  }
});
```

Interactive Video

Interactive Video Viewer is a video player that plays streaming and progressive video encoded in the H.264 format. It also shows interactive product swatches next to the video content. Both single video and Adaptive Video Sets are supported. It is designed to work on both desktop and mobile Web browsers that support HTML5 video. The viewer supports optional closed captions displayed on top of video content, video chapter navigation, and social sharing tools. The purpose of this viewer is to help you implement a "shoppable video" experience. That is, users can select a swatch associated with a particular video time region and get redirected to a Quick View or a product detail page on the customer's website.

Viewer type is 510.

Demo URLs

https://marketing.adobe.com/resources/help/en_US/dm/shoppable-video/glacier/InteractiveVideoViewerDemo.html

and

https://marketing.adobe.com/resources/help/en_US/dm/shoppable-video/AXIS/index.html

System Requirements

See [System requirements](#).

Using Interactive Video Viewer

Interactive Video Viewer represent a main JavaScript file and a set of helper files (a single JavaScript include with all Viewer SDK components used by this particular viewer, assets, and CSS) downloaded by the viewer in runtime.

Interactive Video Viewer can be used in pop-up mode using production-ready HTML page provided with Image Serving Viewers. It can also be used in embedded mode, where it is integrated into the targeted web page using the documented API.

Configuration and skinning is similar to that of the other viewers described in this guide. All skinning is achieved by way of custom (CSS) Cascading Style Sheets.

See [Command reference common to all viewers – Configuration attributes](#) and [Command reference common to all Viewers – URL](#)

Interacting with Interactive Video Viewer

Interactive Video Viewer provides a set of standard user interface controls for video playback, such as a Play/Pause button, video scrubber, video time bubble, played time/total time indicator, volume control, full-screen button, and closed caption toggle. All of these controls are grouped into a control bar directly under the main view.

Note that on touch devices the volume control is hidden from the user interface, because it is only possible to control the volume using the device's hardware buttons.

When the viewer operates in pop-up mode, a full-screen button is not available in the user interface.

The viewer shows a panel with interactive swatches to the right of the video viewing area. The list of swatches automatically advances as the video plays, so that swatches corresponding to the current video region are shown. Clicking or tapping on a swatch triggers an action that was associated with such swatch during author time. Depending on how you set it up, the trigger may redirect to a different page on the web site, or pass product information back to the web page logic, which in turn can trigger the opening of a Quick View that shows related product content.

It is possible to navigate through the video content quickly when video chaptering is activated. Video chapters are displayed as markers in the video scrubber track and show the chapter title and description on roll over (or on a single tap on touch systems). The customer can "seek" to a particular chapter by clicking a chapter marker or tapping a chapter description bubble.

The viewer also supports a variety of social media sharing tools. They are available as a single button in the user interface which expands into a sharing toolbar when the user clicks or taps on it. The sharing toolbar contains an icon for each type of sharing channel supported such as Facebook, Twitter, email share, embed code share, and link share. When email share, embed share, or link share tools are activated, the viewer displays a modal dialog box with a corresponding data entry form. When Facebook or Twitter are called, the viewer redirects the user to a standard sharing dialog box from a social media service. Also, when a sharing tool is activated video playback is paused automatically. Sharing tools are not available in full-screen mode because of web browser security restrictions.

The viewer is fully keyboard accessible. See [Keyboard accessibility and navigation](#).

Embedding Interactive Video Viewer

Interactive Video Viewer is designed to be embedded into the hosting page. Such a web page may have a static layout, or it may be "responsive" and display differently on different devices or for different browser window sizes.

To accommodate these needs, the viewer supports two primary operation modes: fixed size embedding and responsive embedding.

About fixed size embedding mode and responsive design embedding mode

In the embedded mode, the viewer is added to the existing web page, which may already have some customer content not related to the viewer. The viewer normally occupies only a part of a web page's real estate.

The primary use cases are web pages oriented for desktops or tablet devices, and also responsive designed pages that adjust layout automatically depending on the device type.

Fixed size embedding is used when the viewer does not change its size after initial load. This is the best choice for web pages that have a static layout.

Responsive design embedding assumes that the viewer may need to resize at runtime in response to the size change of its container DIV. The most common use case is adding a viewer to a web page that uses a flexible page layout.

In responsive design embedding mode, the viewer behaves differently depending on the way web page sizes its container DIV. If the web page sets only the width of the container DIV, leaving its height unrestricted, the viewer automatically chooses its

height according to the aspect ratio of the asset that is used. This functionality ensures that the asset fits perfectly into the view without any padding on the sides. This use case is the most common for web pages using responsive web design layout frameworks like Bootstrap, Foundation, and so on.

Otherwise, if the web page sets both the width and the height for the viewer's container `DIV`, the viewer fills just that area and follows the size that the web page layout provides. A good example is embedding the viewer into a modal overlay, where the overlay is sized according to web browser window size.

Fixed size embedding

You add the viewer to a web page by doing the following:

1. Adding the viewer JavaScript file to your web page.
 2. Defining the container `DIV`.
 3. Setting the viewer size.
 4. Creating and initializing the viewer.
1. Adding the viewer JavaScript file to your web page.

Creating a viewer requires that you add a script tag in the HTML head. Before you can use the viewer API, be sure that you include `InteractiveVideoViewer.js`. The `InteractiveVideoViewer.js` file is located under the `html5/js/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/etc/dam/viewers/s7viewers/html5/js/InteractiveVideoViewer.js
```

You can use a relative path if the viewer is deployed on one of the Adobe Scene7 servers and it is served from the same domain. Otherwise, you specify a full path to one of Adobe Scene7 servers that have the IS-Viewers installed.

The relative path looks like the following:

```
<script language="javascript" type="text/javascript"
src="/etc/dam/viewers/s7viewers/html5/js/InteractiveVideoViewer.js"></script>
```



Note: You should only reference the main viewer JavaScript `include` file on your page. You should not reference any additional JavaScript files in the web page code which might be downloaded by the viewer's logic in runtime. In particular, do not directly reference `HTML5 SDK Utils.js` library loaded by the viewer from `/s7viewers` context path (so-called consolidated SDK `include`). The reason is that the location of `Utils.js` or similar runtime viewer libraries is fully managed by the viewer's logic and the location changes between viewer releases. Adobe does not keep older versions of secondary viewer `includes` on the server.

As a result, putting a direct reference to any secondary JavaScript `include` used by the viewer on the page breaks the viewer functionality in the future when a new product version is deployed.

2. Defining the container `DIV`.

Add an empty `DIV` element to the page where you want the viewer to appear. The `DIV` element must have its ID defined because this ID is passed later to the viewer API. The `DIV` has its size specified through CSS.

The placeholder `DIV` is a positioned element, meaning that the `position` CSS property is set to `relative` or `absolute`.

For the full-screen feature to function properly in Internet Explorer make sure that there are no other elements in the DOM that have a higher stacking order than your placeholder `DIV`.

The following is an example of a defined placeholder `DIV` element:

```
<div id="s7viewer" style="position:relative"></div>
```

3. Setting the viewer size

You can set the static size for the viewer by either declaring it for `.s7interactivevideoviewer` top-level CSS class in absolute units, or by using `stagesize` modifier.

You can put sizing in CSS directly on the HTML page, or in a custom viewer CSS file, which is then later assigned to a viewer preset record in AEM Assets – on-demand, or passed explicitly using `style` command.

See [Customizing Interactive Video Viewer](#) for more information about styling the viewer with CSS.

The following is an example of defining a static viewer size in the HTML page:

```
#s7viewer.s7interactivevideoviewer {
  width: 640px;
  height: 640px;
}
```

You can set the `stagesize` modifier in the viewer preset record in AEM Assets - on-demand. Or, you can pass it explicitly with the viewer initialization code with `params` collection, or as an API call as described in the Command Reference section, like this:

```
interactivevideoviewer.setParam("stagesize", "640,640");
```

A CSS-based approach is recommended and is used in this example.

4. Creating and initializing the viewer.

When you have completed the steps above, you create an instance of `s7viewers.InteractiveVideoViewer` class, pass all configuration information to its constructor, and call `init()` method on a viewer instance. Configuration information is passed to the constructor as a JSON object. At minimum, this object should have `containerId` field which holds the name of the viewer container ID and nested `params` JSON object with configuration parameters supported by the viewer.

In this case, the `params` object must have at least the Image Serving URL passed as `serverUrl` property, and the initial asset as `asset` parameter. The JSON-based initialization API lets you create and start the viewer with a single line of code, video server URL passed as `videoserverurl` property, initial asset as `asset` parameter and interactive data as `interactivedata` property. JSON-based initialization API lets you create and start the viewer with single line of code.

It is important to have the viewer container added to the DOM so that the viewer code can find the container element by its ID. Some browsers delay building DOM until the end of the web page. For maximum compatibility, call the `init()` method just before the closing BODY tag, or on the `body onload()` event.

At the same, the container element should not necessarily be part of the web page layout just yet. For example, it may be hidden using `display:none` style assigned to it. In this case, the viewer delays its initialization process until the moment when the web page brings the container element back to the layout. What that happens, the viewer load automatically resumes.

The following is an example of creating a viewer instance, passing the minimum necessary configuration options to the constructor and calling the `init()` method. The example assumes the following:

- The viewer instance is `interactiveVideoViewer`.
- The name of placeholder DIV is `s7viewer`.
- The Image Serving URL is `https://aodmarketingna.assetsadobe.com/is/image/`.
- The video server URL is `https://gateway-na.assetsadobe.com/DMGateway/public/aodmarketingna`.
- The content URL is `https://aodmarketingna.assetsadobe.com/`.
- The asset is `/content/dam/mac/aodmarketingna/dm-viewers-content/video/Glacier.mp4`.

- The interactive data is

```
is/content/content/dam/mac/aodmarketingna/_VTT/dm-viewers-content/video/Glacier.mp4.svideo.vtt.
```

```
<script type="text/javascript">
var interactiveVideoViewer = new s7viewers.InteractiveVideoViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "/content/dam/mac/aodmarketingna/dm-viewers-content/video/Glacier.mp4",
    "config": "/etc/dam/presets/viewer/Shoppable_Video_Dark",
    "serverurl": "https://aodmarketingna.assetsadobe.com/is/image/",
    "videoserverurl": "https://gateway-na.assetsadobe.com/DMGateway/public/aodmarketingna",
    "contenturl": "https://aodmarketingna.assetsadobe.com/",
    "interactivedata": "is/content/content/dam/mac/aodmarketingna/_VTT/dm-viewers-content/video/Glacier.mp4.svideo.vtt"
  }
}).init();
</script>
```

The following code is a complete example of a trivial web page that embeds the Video Video Viewer with a fixed size:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="https://aodmarketingna.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/InteractiveVideoViewer.js"></script>
<style type="text/css">
#s7viewer.s7interactivevideoviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative;"></div>
<script type="text/javascript">
var interactiveVideoViewer = new s7viewers.InteractiveVideoViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "/content/dam/mac/aodmarketingna/dm-viewers-content/video/Glacier.mp4",
    "config": "/etc/dam/presets/viewer/Shoppable_Video_Dark",
    "serverurl": "https://aodmarketingna.assetsadobe.com/is/image/",
    "videoserverurl": "https://gateway-na.assetsadobe.com/DMGateway/public/aodmarketingna",
    "contenturl": "https://aodmarketingna.assetsadobe.com/",
    "interactivedata": "is/content/content/dam/mac/aodmarketingna/_VIT/dm-viewers-content/video/Glacier.mp4.svideo.vtt"
  }
}).init();
</script>
</body>
</html>
```

Responsive design embedding with unrestricted height

With responsive design embedding, the web page normally has some kind of flexible layout in place that dictates the runtime size of the viewer's container DIV. For the following example, assume that the web page allows the viewer's container DIV to take 40% of the web browser window size, leaving its height unrestricted. The web page HTML code would look like the following:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
.holder {
  width: 40%;
}
</style>
</head>
<body>
<div class="holder"></div>
```

```
</body>
</html>
```

Adding the viewer to such a page is similar to the steps for fixed size embedding. The only difference is that you do not need to explicitly define the viewer size.

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Creating and initializing the viewer.

All the steps above are the same as with the fixed size embedding. Add the container DIV to the existing "holder" DIV. The following code is a complete example. Notice how viewer size changes when the browser is resized, and how the viewer aspect ratio matches the asset.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="https://aodmarketingna.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/InteractiveVideoViewer.js"></script>
<style type="text/css">
.holder {
width: 40%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var interactiveVideoViewer = new s7viewers.InteractiveVideoViewer({
  "containerId":"s7viewer",
"params":{
  "asset":"/content/dam/mac/aodmarketingna/dm-viewers-content/video/Glacier.mp4",
"config":"/etc/dam/presets/viewer/Shoppable_Video_Dark",
"serverurl":"https://aodmarketingna.assetsadobe.com/is/image/",
  "videoserverurl":"https://gateway-na.assetsadobe.com/DMGateway/public/aodmarketingna",
  "contenturl":"https://aodmarketingna.assetsadobe.com/",
"interactivedata":"is/content/content/dam/mac/aodmarketingna/_VIT/dm-viewers-content/video/Glacier.mp4.svideo.vtt"
}
}).init();
</script>
</body>
</html>
```

The following examples page illustrates more real-life uses of responsive design embedding with unrestricted height:

https://marketing.adobe.com/resources/help/en_US/s7/vlist/vlist.html

Responsive Embedding with Width and Height Defined

In case of responsive embedding with width and height defined, the web page styling is different. It provides both sizes to the "holder" DIV and center it in the browser window. Also, the web page sets the size of the HTML and BODY element to 100 percent.

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
width: 100%;
height: 100%;
}
.holder {
```

```

position: absolute;
left: 20%;
top: 20%;
width: 60%;
height: 60%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>

```

The rest of the embedding steps are identical to the steps used for responsive embedding with unrestricted height. The resulting example is the following:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="https://aodmarketingna.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/InteractiveVideoViewer.js"></script>
<style type="text/css">
html, body {
width: 100%;
height: 100%;
}
.holder {
position: absolute;
left: 20%;
top: 20%;
width: 60%;
height: 60%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var interactiveVideoViewer = new s7viewers.InteractiveVideoViewer({
"containerId":"s7viewer",
"params":{
"asset":"/content/dam/mac/aodmarketingna/dm-viewers-content/video/Glacier.mp4",
"config":"/etc/dam/presets/viewer/Shoppable_Video_Dark",
"serverurl":"https://aodmarketingna.assetsadobe.com/is/image/",
"videoserverurl":"https://gateway-na.assetsadobe.com/DMGateway/public/aodmarketingna",
"contenturl":"https://aodmarketingna.assetsadobe.com/",
"interactivedata":"is/content/content/dam/mac/aodmarketingna/_VIT/dm-viewers-content/video/Glacier.mp4.svideo.vtt"
}
}).init();
</script>
</body>
</html>

```

Embedding Using Setter-based API

Instead of using JSON-based initialization, it is possible to use setter-based API and no-args constructor. Using this API constructor does not take any parameters and configuration parameters are specified using `setContainerId()`, `setParam()`, and `setAsset()` API methods with separate JavaScript calls.

The following example illustrates using fixed size embedding with the setter-based API:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="https://aodmarketingna.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/InteractiveVideoViewer.js"></script>

```

```
<style type="text/css">
#s7viewer.s7interactivevideoviewer {
  width: 640px;
  height: 480px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative;width:640px;height:360px;"></div>
<script type="text/javascript">
var interactiveVideoViewer = new s7viewers.InteractiveVideoViewer();
interactiveVideoViewer.setContainerId("s7viewer");
interactiveVideoViewer.setParam("config", "/etc/dam/presets/viewer/Shoppable_Video_Dark");
interactiveVideoViewer.setParam("serverurl", "https://aodmarketingna.assetsadobe.com/is/image/");
interactiveVideoViewer.setParam("videoserverurl",
"https://gateway-na.assetsadobe.com/DMGateway/public/aodmarketingna");
interactiveVideoViewer.setParam("contenturl", "https://aodmarketingna.assetsadobe.com/");
interactiveVideoViewer.setParam("interactivedata",
"is/content/content/dam/mac/aodmarketingna/_VTT/dm-viewers-content/video/Glacier.mp4.svideo.vtt");
interactiveVideoViewer.setAsset("/content/dam/mac/aodmarketingna/dm-viewers-content/video/Glacier.mp4");
interactiveVideoViewer.init();
</script>
</body>
</html>
```

Command reference – Configuration attributes

Configuration attributes documentation for Interactive Video Viewer.

Any configuration command can be set in URL or using `setParam()`, or `setParams()`, or both, API methods. Any config attribute can be also specified in the server-side configuration record.

Some configuration commands may be prefixed with the class name or instance name of corresponding Viewer SDK component. An instance name of the component is dynamic and depends on the ID of the viewer container DOM element passed to `setContainerId()` API method. Documentation includes an optional prefix for such commands. For example, `playback` command is documented as follows:

`[VideoPlayer.|<containerId>_videoPlayer].playback`

which means that you can use this command as:

- `playback` (short syntax)
- `VideoPlayer.playback` (qualified with component class name)
- `cont_videoPlayer.playback` (qualified with component ID, assuming `cont` is the ID of the container element)

See also [Command reference common to all viewers – Configuration attributes](#)

CallToAction.align

Configuration attribute for Video Video Viewer.

`[CallToAction.|<containerId>_callToAction.]align=left|center|right`

left center right	<p>Specifies the internal horizontal alignment (or anchoring) of the thumbnails container within the component area.</p> <p>In call-to-action, the internal thumbnail container is sized so that only a whole number of thumbnails is shown. As a result, there is some padding between the internal container and the external component bounds.</p>
-------------------	---

	This modifier specifies how the internal thumbnails container is positioned horizontally inside the component.
--	--

Properties

Optional.

Default

center

Example

```
align=left
```

CallToAction.direction

Configuration attribute for Video Video Viewer.

```
[CallToAction. | <containerId>_callToAction. ]direction=auto|left|right
```

auto left right	<p>Specifies the way thumbnails fill in the view.</p> <p>Set to <code>left</code> to set the left-to-right fill order.</p> <p>Set to <code>right</code> reverses the order so that the view is filled in a right-to-left, top-to-bottom direction.</p> <p>Set to <code>auto</code> to have the component apply right mode when the locale is set to " ja "; otherwise, <code>left</code> is used.</p>
-----------------	---

Properties

Optional.

Default

auto

Example

```
direction=right
```

CallToAction.enabledragging

Configuration attribute for Video Video Viewer.

```
[CallToAction. | <containerId>_callToAction. ]enabledragging=0|1[ ,overdragvalue]
```

0 1	Enables or disables the ability for a user to scroll the thumbnails with a mouse or by using touch gestures.
-----	--

<code>overdragvalue</code>	<p>Is in the 0–1 range and it is a percent value for the movement in the wrong direction of the actual speed.</p> <p>If set to 1 it moves with the mouse.</p> <p>If set to 0 it does not let you move in the wrong direction.</p>
----------------------------	---

Properties

Optional.

Default

1, 0.5

Example

```
enabledragging=0
```

CallToAction.fmt

Configuration attribute for Video Video Viewer.

```
[CallToAction. | <containerId>_callToAction. ]fmt=jpg | jpeg | png | png-alpha | gif | gif-alpha
```

<code>jpg jpeg png png-alpha gif gif-alpha</code>	<p>Specifies the image format that the component uses for loading images from Image Server.</p> <p>If the specified format ends with "-alpha", the component renders the images as transparent content. For all other image formats the component treats the images as opaque.</p>
---	--

Properties

Optional.

Default

jpeg

Example

```
fmt=png-alpha
```

CallToAction.maxloadradius

Configuration attribute for Video Video Viewer.

```
[CallToAction. | <containerId>_callToAction. ]maxloadradius=-1 | 0 | preloadnbr
```

<code>-1 0 preloadnbr</code>	<p>Specifies the component preload behavior.</p> <p>When set to -1 all the thumbnails are loaded simultaneously when the component is initialized or the asset is changed.</p> <p>When set to 0 only visible thumbnails are loaded.</p>
----------------------------------	---

	Set to <i>preloadnbr</i> to define how many invisible rows/columns around the visible area are preloaded.
--	---

Properties

Optional.

Default

-1

Example

```
maxloadradius=-1
```

CallToAction.scrollbar

Configuration attribute for Video Video Viewer.

[CallToAction. | <containerId>_callToAction.]scrollbar=0 | 1

0 1	Enable the use of the scroll bar.
-------	-----------------------------------

Properties

Optional.

Default

1

Example

```
scrollbar=0
```

CallToAction.textpos

Configuration attribute for Video Video Viewer.

[CallToAction. | <containerId>_callToAction.]textpos=bottom | top | left | right | none | tooltip

bottom top left right none tooltip	<p>Specifies where the label is drawn relative to the thumbnail image. That is, the label is centered at the specified location relative to the thumbnail.</p> <p>When <i>tooltip</i> is specified, the label text is displayed as a floating tooltip over the thumbnail image.</p> <p>Set to <i>none</i> to turn off the label.</p>
------------------------------------	--

Properties

Optional.

Default

bottom

Example

```
textpos=top
```

callToActionRecap

Configuration attribute for Video Video Viewer.

callToActionRecap=0|1

0 1	Indicates whether the viewer shows the "call-to-action" panel at the conclusion of video playback.
-----	--

Properties

Optional.

Default

1

Example

```
callToActionRecap=0
```

ControlBar.transition

Configuration attribute for Video Video Viewer.

[ControlBar.<containerId>_controls.]transition=none|fade[,delaytohide[,duration]]

none fade	Specifies the effect type which will be used to show/hide the control bar and its content. Set to none for instant show/hide. Set to fade to provide a gradual fade in/out effect. Not supported on Internet Explorer 8.
delaytohide	Specifies the time in seconds between the last mouse/touch event registered by the control bar and the time control bar hides. If set to -1 the component never triggers its auto-hide effect and, therefore, always stays visible on the screen.
duration	Sets the duration of the fade in/out animation, in seconds.

Properties

Optional.

Default

fade,2,0.5

Example

```
transition=none
```

InteractiveSwatches.autoscroll

Configuration attribute for Video Video Viewer.

```
[InteractiveSwatches.<containerId>_interactiveSwatches.]autoscroll=0|1
```

0 1	Enables or disables the ability for interactive swatches to scroll automatically along with each media transition, such as during video playback.
-----	---

Properties

Optional.

Default

1

Example

```
autoscroll=0
```

InteractiveSwatches.direction

Configuration attribute for Video Video Viewer.

```
[InteractiveSwatches.<containerId>_interactiveSwatches.]direction=auto|left|right
```

auto left right	<p>Specifies the way swatches fill in the view.</p> <p>Set to <code>left</code> to set the left-to-right fill order.</p> <p>Set to <code>right</code> reverses the order so that the view is filled in a right-to-left, top-to-bottom direction.</p> <p>When <code>auto</code> is set, the component applies right mode when locale is set to "ja"; otherwise, <code>left</code> is used.</p>
-----------------	---

Properties

Optional.

Default

auto

Example

```
direction=right
```

InteractiveSwatches.displaymode

Configuration attribute for Video Video Viewer.

[InteractiveSwatches.<containerId>_interactiveSwatches.]displaymode=continuous|segment

continuous segment	Populates interactive swatches continuously with thumbnails regardless of segment boundaries or, allows the empty space to note the segment boundaries.
--------------------	---

Properties

Optional.

Default

segment

Example

displaymode=continuous

InteractiveSwatches.enabledragging

Configuration attribute for Video Video Viewer.

[InteractiveSwatches.<containerId>_interactiveSwatches.]enabledragging=0|1[,overdragvalue]

0 1	Enables or disables the ability for a user to scroll the swatches with a mouse or by using touch gestures.
overdragvalue	<p>Is in the 0–1 range and it is a percent value for the movement in the wrong direction of the actual speed.</p> <p>If set to 1 it moves with the mouse.</p> <p>If set to 0 it does not let you move in the wrong direction.</p>

Properties

Optional.

Default

1,0.5

Example

enabledragging=0

InteractiveSwatches.fmt

Configuration attribute for Video Video Viewer.

[InteractiveSwatches.<containerId>_interactiveSwatches.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha

jpg jpeg png png-alpha gif gif-alpha	Specifies the image format that the component uses for loading images from Image Server.
--------------------------------------	--

	If the specified format ends with "-alpha", the component renders the images as transparent content. For all other image formats the component treats the images as opaque. Note that the component has a white background by default. Therefore, to make it completely transparent set the background-color CSS property to transparent.
--	---

Properties

Optional.

Default

jpeg

Example

```
fmt=png-alpha
```

InteractiveSwatches.maxloadradius

Configuration attribute for Video Video Viewer.

[InteractiveSwatches.<containerId>_interactiveSwatches.]maxloadradius=-1|0|preloadnbr

1 0 preloadnbr	<p>Specifies the component preload behavior.</p> <p>When set to -1 all swatches are loaded simultaneously when the component is initialized or the asset is changed.</p> <p>When set to 0 only visible swatches are loaded.</p> <p>Set to preloadnbr to define how many invisible rows/columns around the visible area are preloaded.</p>
----------------	---

Properties

Optional.

Default

1

Example

```
maxloadradius=-1
```

InteractiveSwatches.scrollstep

Configuration attribute for Video Video Viewer.

[InteractiveSwatches.<containerId>_interactiveSwatches.]scrollstep=step

step	Specifies the number of swatches to scroll for each tap of the corresponding scroll button.
------	---

	If the specified value is greater than the number of interactive swatches visible, each tap only scrolls by the number of visible swatches to prevent the omission of any swatch.
--	---

Properties

Optional.

Default

3

Example

`scrollstep=1`

InteractiveSwatches.textpos

Configuration attribute for Video Video Viewer.

[InteractiveSwatches. |<containerId>_interactiveSwatches.]textpos=bottom|top|left|right|none|tooltip

bottom top left right none tooltip	<p>Specifies where the label is drawn relative to the swatch image. That is, the label is centered at the specified location relative to the thumbnail.</p> <p>When <code>tooltip</code> is specified, the label text is displayed as a floating tooltip over the thumbnail image.</p> <p>Set to <code>none</code> to turn off the label.</p>
------------------------------------	---

Properties

Optional.

Default

bottom

Example

`textpos=top`

SocialShare.bearing

Configuration attribute for Video Video Viewer.

[SocialShare. |<containerId>_socialShare.]bearing=up|down|left|right|fit-vertical|fit-lateral

up down left right fit-vertical fit-lateral	<p>Specifies the direction of slide animation for buttons container. When set to <code>up</code>, <code>down</code>, <code>left</code>, or <code>right</code>, the panel rolls out in the specified direction without any additional bounds checking, which may result in the panel being clipped by an outside container.</p> <p>When set to <code>fit-vertical</code>, the component first shifts the base panel position to the bottom of <code>SocialShare</code> and tries to roll out the panel in one of the following directions from such a base location: <code>bottom</code>, <code>right</code>, <code>left</code>. With each attempt the component checks if panel is clipped</p>
---	--

	<p>by an outside container. If all attempts fail, the component tries to shift the base panel position to the top and repeats the roll out attempts from a top, right, and left direction.</p> <p>When set to <code>fit-lateral</code>, the component uses a similar logic, but shifts the base to the right first, ttrying right, down, and up rollout directions. Then, it shifts the base to the left, trying left, down and up rollout directions.</p>
--	--

Properties

Optional.

Default

`fit-vertical`

Example

`bearing=left`

VideoPlayer.autoplay

Configuration attribute for Video Video Viewer.

`[VideoPlayer.<containerId>_videoPlayer.]autoplay=0|1`

<code>0 1</code>	Indicates whether the viewer starts playing the video on load. Note that some systems—such as certain mobile devices—may not support autoplay.
------------------	--

Properties

Optional.

Default

`0`

Example

`autoplay=1`

VideoPlayer.iconeffect

Configuration attribute for Video Video Viewer.

`[VideoPlayer.<containerId>_videoPlayer.]iconeffect=0|1[,count][,fade][,autoHide]`

<code>0 1</code>	Enables the IconEffect to be displayed on top of the video when the video is in a paused state. On some devices native controls are used. In such cases, the <code>iconeffect</code> modifier is ignored.
<code>count</code>	Specifies the maximum number of times the IconEffect appears and reappears. A value of <code>-1</code> indicates that the icon reappears indefinitely.

<i>fade</i>	Specifies the duration of show or hide animation, in seconds.
<i>autoHide</i>	Sets the number of seconds that the IconEffect stays fully visible before it auto-hides. That is, the time after the fade in animation is completed and before the fade out animation starts. Set to 0 to disable auto-hide behavior.

Properties

Optional.

Default

1,-1,0.3,0

Example

```
iconeffect=0
```

VideoPlayer.initialbitrate

Configuration attribute for Video Video Viewer.

```
[VideoPlayer.<containerId>_videoPlayer.]initialbitrate=value
```

value	<p>Sets the video bitrate in kbits per seconds (or kbps) that is used for initial playback of video on desktop systems. If this bit rate value does not exist in the Adaptive Video Set then the video player starts with the video that has the next lower bitrate.</p> <p>If set to 0 the video player starts from the lowest possible bitrate. Applicable only for Flash-enabled desktop systems when playback mode is set to auto.</p>
-------	--

Properties

Optional.

Default

1400

Example

```
initialbitrate=600
```

VideoPlayer.loop

Configuration attribute for Video Video Viewer.

```
[VideoPlayer.<containerId>_videoPlayer.]loop=0|1
```

0 1	Indicates whether the video should play again after playback is complete.
-----	---

Properties

Optional.

Default

0

Example

```
loop=1
```

VideoPlayer.playback

Configuration attribute for Video Video Viewer.

```
[VideoPlayer.<containerId>_videoPlayer.] playback=auto|progressive
```

auto progressive	<p>Sets the type of playback used by the viewer.</p> <p>When <code>auto</code> is set, on most desktop browsers, the component uses flash video playback if Flash Player is installed, and HTML5 playback otherwise.</p> <p>If <code>progressive</code> is specified, the component relies on HTML5 playback exclusively as natively supported by browsers and will play video progressively on all systems.</p> <p>For more information on the playback selection in <code>auto</code> and <code>progressive</code> modes, see the Viewer SDK User Guide.</p>
------------------	--

Properties

Optional.

Default

auto

Example

```
playback=progressive
```

VideoPlayer.posterimage

Configuration attribute for Video Video Viewer.

```
[VideoPlayer.<containerId>_videoPlayer.] posterimage=none|[image_id][?isCommands]
```

none [image_id][?isCommands]	<p>The image to display on the first frame before the video starts playing, resolved against <code>serverurl</code>. If specified in the URL, HTTP-encode the following:</p> <ul style="list-style-type: none">• <code>?</code> as <code>%3F</code>• <code>&</code> as <code>%26</code>• <code>=</code> as <code>%3D</code>
------------------------------	---

	<p>If the <i>image_id</i> value is omitted, the component attempts to use the default poster image for that asset instead.</p> <p>When the video is specified as a path, the default poster images catalog id is derived from the video path as the <i>catalog_id/image_id</i> pair where <i>catalog_id</i> corresponds to the first token in the path and <i>image_id</i> is the name of the video with the extension removed. If the image with that ID does not exist, the poster image is not shown.</p> <p>To prevent the display of the default poster image, specify <i>none</i> as the poster image value. If only the <i>isCommands</i> are specified the commands are applied to the default poster image before the image is displayed.</p>
--	--

Properties

Optional.

Default

None.

Example

```
posterimage=none
```

VideoPlayer.progressivebitrate

Configuration attribute for Video Video Viewer.

```
[VideoPlayer.|<containerId>_videoPlayer.]progressivebitrate=value
```

value	<p>Specifies in kbits per seconds (or kbps), the desired video bit rate to play from an Adaptive Video Set in case the current system does not support adaptive video playback.</p> <p>The component picks up the video stream with the closest possible (but not exceeding) bit rate to the specified value. If all video streams in the Adaptive Video Set have higher quality than the specified value, the logic chooses the bit rate with the lowest quality.</p>
-------	--

Properties

Optional.

Default

700

Example

```
progressivebitrate=600
```

VideoPlayer.singleclick

Configuration attribute for Video Video Viewer.

```
[VideoPlayer.|<containerId>_videoPlayer.]singleclick=none|playPause
```

none playPause	Configures the mapping of single-click/tap to toggle play/pause. Setting to none disables single-click/tap to play/pause. If set to playPause then clicking on the video toggles between playing and pausing the video. On some devices you can use native controls. In this case, a singleclick behavior is disabled.
------------------	--

Properties

Optional.

Default

playPause

Example

```
singleclick=none
```

VideoPlayer.smoothing

Configuration attribute for Video Video Viewer.

```
[VideoPlayer. |<containerId>_videoPlayer.]smoothing=0 | 1
```

0 1	<p>Specifies whether the video is smoothed (interpolated) when it is scaled. For smoothing to work, the runtime must be in high quality mode—the default.</p> <p>The default value is 0 (no smoothing). Set this property to 1 to take advantage of mipmapping image optimization. Only for Flash playback.</p>
-------	---

Properties

Optional.

Default

0

Example

```
smoothing=1
```

VideoPlayer.ssl

Configuration attribute for Video Video Viewer.



Note: This configuration attribute only applies to AEM 6.2 with installation of [Feature Pack NPR-13480](#) and to AEM 6.1 with installation of [Feature Pack NPR-15011](#).

```
[VideoPlayer. |<containerId>_videoPlayer.]ssl=auto | on
```

auto on	Controls whether the video is delivered over a secure SSL connection (HTTPS) or an insecure connection(HTTP).
-----------	---

	<p>When set to <code>auto</code> the video delivery protocol is inherited from the protocol of the embedding web page. If the web page is loaded over HTTPS, the video is also delivered over HTTPS, and vice versa. If the web page is on HTTP, the video is delivered over HTTP.</p> <p>When set to <code>on</code>, video delivery always occurs over a secure connection without regard to the web page protocol.</p> <p>Affects published video delivery only and is ignored for video preview in Author mode.</p>
--	---

Properties

Optional.

Default

`auto`

Example

```
ssl=on
```

See also [Secure Video Delivery](#).

VideoPlayer.waiticon

Configuration attribute for Video Video Viewer.

```
[VideoPlayer.<containerId>_videoPlayer.]waiticon=0|1
```

0 1	Enables or disables buffering animation (wait icon) display.
-----	--

Properties

Optional.

Default

`1`

Example

```
waiticon=0
```

VideoScrubber.chaptertimepattern

Configuration attribute for Video Video Viewer.

```
[VideoScrubber.<containerId>_videoScrubber.]chaptertimepattern=[h:]m|mm:s|ss
```

[h:]m mm:s ss	Sets the pattern for the time that is displayed in the title bar of the chapter lable, where <code>h</code> represents hours, <code>m</code> for minutes, and <code>s</code> for seconds.
---------------	---

	<p>The number of letters used for each time unit determines the number of digits to display for the unit. If the number cannot fit into the given digits, the equivalent value is displayed in the subsequent unit.</p> <p>For example, if the current movie time is 67 minutes and 5 seconds, a time pattern of <code>m:ss</code> displays as 67:05. The same time is displayed as 1:07:5 if the time pattern is <code>h:mm:s</code>.</p>
--	--

Properties

Optional.

Default

`m:ss`

Example

```
chaptertimepattern=h:mm:ss
```

VideoScrubber.showchaptertime

Configuration attribute for Video Video Viewer.

`[VideoScrubber.<containerId>_videoScrubber.]showchaptertime=0|1`

0 1	Enables/disables the chapter time in the title bar of the chapter label.
-----	--

Properties

Optional.

Default

1

Example

```
showchaptertime=0
```

VideoScrubber.showchaptertitle

Configuration attribute for Video Video Viewer.

`[VideoScrubber.<containerId>_videoScrubber.]showchaptertitle=0|1`

0 1	Enables/disables the title bar of the chapter label. The chapter start time is not displayed when the title bar is disabled.
-----	--

Properties

Optional.

Default

1

Example

```
showchaptertitle=0
```

VideoScrubber.timepattern

Configuration attribute for Video Video Viewer.

```
[VideoScrubber. |<containerId>_videoScrubber. ]timepattern=[h:]m|mm:s|ss
```

[h:]m mm:s ss	<p>Sets the pattern for the time that is displayed in the time bubble, where <i>h</i> represents hours, <i>m</i> for minutes, and <i>s</i> for seconds.</p> <p>The number of letters used for each time unit determines the number of digits to display for the unit. If the number cannot fit into the given digits, the equivalent value is displayed in the subsequent unit.</p> <p>For example, if the current movie time is 67 minutes and 5 seconds, a time pattern of <i>m:ss</i> displays as 67:05. The same time is displayed as 1:07:5 if the time pattern is <i>h:mm:s</i>.</p>
---------------	--

Properties

Optional.

Default

mm:s

Example

```
timepattern=h:mm:ss
```

VideoTime.timepattern

Configuration attribute for Video Video Viewer.

```
[VideoTime. |<containerId>_videoTime. ]timepattern=[h:]m|mm:s|ss
```

[h:]m mm:s ss	<p>Sets the pattern for the time that is displayed in the control bar, where <i>h</i> represents hours, <i>m</i> for minutes, and <i>s</i> for seconds.</p> <p>The number of letters used for each time unit determines the number of digits to display for the unit. If the number cannot fit into the given digits, the equivalent value is displayed in the subsequent unit.</p> <p>For example, if the current movie time is 67 minutes and 5 seconds, a time pattern of <i>m:ss</i> displays as 67:05. The same time is displayed as 1:07:5 if the time pattern is <i>h:mm:s</i>.</p>
---------------	--

Properties

Optional.

Default

m:ss

Example

```
timepattern=h:mm:ss
```

Command reference – URL

Command reference documentation for Video Video Viewer.

You can set any configuration command in the URL. Or, you can use the API methods `setParam()`, or `setParams()`, or both to set any configuration command. You can also specify any config attribute in the server-side configuration record.

You can prefix some configuration commands with the class name or the instance name of corresponding Viewer SDK component. An instance name of the component is dynamic and depends on the ID of the viewer container DOM element passed to `setContainerId()` API method. Documentation includes optional prefixes for such commands. For example, `playback` is documented as follows:

```
[VideoPlayer.<containerId>_videoPlayer].playback
```

which means that this command is used in the following manner:

- `playback` (short syntax)
- `VideoPlayer.playback` (qualified with component class name)
- `cont_videoPlayer.playback` (qualified with component ID, assuming that `cont` is the ID of the container element)

See also [Command reference common to all viewers – Configuration attributes](#).

See also [Command reference common to all Viewers – URL](#).

caption

URL command for Video Video Viewer.

```
caption=file[,0|1]
```

The viewer supports closed captioning through hosted WebVTT files. Overlapping cues and regions are not supported. Supported cue positioning operators include the following:

Key	Name	Value	Description
A	text align	left right middle start end	Control text alignment. Default is middle.
T	text position	0%–100%	Percentage of inset into the VideoPlayer component for the beginning of the caption text. Default is 0%.
S	line size	0%–100%	Percentage of video width used for captions. Default is 100%.

Key	Name	Value	Description
L	line position	0%–100% integer	<p>Determines the line position on the page.</p> <p>If it is expressed as an integer (no percent sign), then it is the number of lines from the top where the text is displayed.</p> <p>If it is a percentage (the percent sign is the last character), then the caption text is displayed that percent down the display area.</p> <p>Default is 100%.</p>

Other WebVTT features present in the WebVTT file are not supported, but should not disrupt the captioning.

<i>file</i>	Specifies a URL or path to the WebVTT caption content. Serve the WebVTT file by Image Serving.
0 1	Specifies the default caption state (enabled is 1).

Properties

Optional.

Default

None.

Example

```
caption=is/content/content/dam/mac/aodmarketingna/_VTT/dm-viewers-content/video/Glacier.mp4.caption.vtt,1
```

interactivedata

URL command for Video Video Viewer.

```
interactivedata=file
```

Interactive data associates certain time regions in the video content with the product data that is later displayed in interactive swatches next to the video. It is also associated in the call-to-action panel shown at the conclusion of video playback. It also provides a title for the interactive video displayed in the call-to-action panel.

<i>file</i>	Specifies a URL or path to WebVTT interactive data content. The WebVTT file must be served by Image Serving.
-------------	--

Properties

Optional.

Default

None.

Example

```
interactivedata=is/content/content/dam/mac/aodmarketingna/_VTT/dm-viewers-content/video/Glacier.mp4.svideo.vtt
```

navigation

URL command for Video Video Viewer.

```
navigation=file
```

The viewer supports video chapter navigation by way of hosted WebVTT files. Cue positioning operators are not supported.

<i>file</i>	Specifies a URL or path to WebVTT navigation content. Image Serving should host the WebVTT file.
-------------	--

Properties

Optional.

Default

None.

Example

```
navigation=is/content/content/dam/mac/aodmarketingna/_VTT/dm-viewers-content/video/Glacier.mp4.navigation.vtt,1
```

videoServerUrl

URL command for Video Viewer.

```
videoServerUrl=videoRootPath
```

<i>videoRootPath</i>	The video server root path. If no domain is specified, the domain from which the page is served is applied instead. Standard URI path resolution applies.
----------------------	---

Properties

Optional.

Default

```
/is/content/
```

Example

```
videoServerUrl=https://gateway-na.assetsadobe.com/DMGateway/public/aodmarketingna
```

JavaScript API reference for Interactive Image Viewer

The main class of the Interactive Image Viewer is `InteractiveVideoViewer`. It is declared in the `s7viewers` namespace. This JavaScript API covers constructor, methods, and callbacks of this particular class.

In all the following examples, `<instance>` stands for the actual name of the JavaScript viewer object that is instantiated from the `s7viewers.InteractiveVideoViewer` class.

dispose

JavaScript API reference for Video Video Viewer.

```
dispose()
```

Disposes this viewer instance by releasing all resources used by the viewer logic and deleting all inner objects and components created by the viewer in runtime.

The web page code should also delete the viewer instance variable as well to completely remove the viewer from the web browser memory.

If the web page code has registered event listeners directly on Viewer SDK components used by the viewer—or stored external references to such components—such listeners must be explicitly unregistered by the web page code, and such external component references must be deleted prior to calling `dispose()`.

Do not access the Viewer API any more after `dispose()` is called.

Parameters

None.

Returns

None.

Example

```
<instance>.dispose()
```

getComponent

JavaScript API reference for Video Video Viewer.

```
getComponent(componentId)
```

Returns a reference to the Viewer SDK component that is used by the viewer. The web page can use this method to extend or customize the behavior of the out-of-box viewer. Call this method only after the `initComplete` viewer callback has run, otherwise the component may not be created yet by the viewer logic.

Parameters

componentID – {String} an ID of the Viewer SDK component used by the viewer. This viewer supports the following component IDs:

Component ID	Viewer SDK component class name
parameterManager	s7sdk.ParameterManager
container	s7sdk.common.Container
mediaSet	s7sdk.set.MediaSet

Component ID	Viewer SDK component class name
videoPlayer	s7sdk.video.VideoPlayer
interactiveSwatches	s7sdk.set.InteractiveSwatches
callToAction	s7sdk.set.CallToAction
fullScreenButton	s7sdk.common.FullScreenButton
mutableVolume	s7sdk.video.MutableVolume
playPauseButton	s7sdk.common.PlayPauseButton
videoScrubber	s7sdk.video.VideoScrubber
videoTime	s7sdk.video.VideoTime
closedCaptionButton	s7sdk.common.ClosedCaptionButton
controls	s7sdk.common.ControlBar
socialShare	s7sdk.share.SocialShare
twitterShare	s7sdk.share.TwitterShare
facebookShare	s7sdk.share.FacebookShare
linkShare	s7sdk.share.LinkShare

When working with SDK APIs it is important to use correct fully qualified SDK namespace as described in [Viewer SDK namespace](#).

See the *Viewer SDK API* documentation for more information about a particular component.

Returns

{Object} a reference to Viewer SDK component. The method returns null if the `componentId` is not a supported viewer component or if the component was not yet created by the viewer logic.

Example

```
<instance>.setHandlers({  
  "initComplete":function() {  
    var videoPlayer = <instance>.getComponent("videoPlayer");  
  }  
})
```

init

JavaScript API reference for Video Video Viewer.

`init()`

Starts the initialization of the Video Video Viewer. By this time the container DOM element must be created so that the viewer code can find it by its ID.

If the container element is not a part of the web page layout just yet (for example, it may be hidden using `display:none` style assigned to it), the viewer suspends its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load automatically resumes.

Call this method only once during viewer life cycle; subsequent calls are ignored.

Parameters

None.

Returns

{Object} A reference to the viewer instance.

Example

```
<instance>.init()
```

InteractiveVideoViewer

JavaScript API reference for Video Video Viewer.

`InteractiveVideoViewer([config])`

Constructor, creates a new Video Image Viewer instance.

Parameters

<i>config</i>	<p>{object} optional JSON configuration object, allows all the viewer settings to pass to the constructor to avoid calling individual setter methods. Contains the following properties:</p> <ul style="list-style-type: none">• <code>containerId</code> – {String} ID of the DOM container (normally a <code>DIV</code>) into which the viewer is inserted. By the time this method is called, it is not necessary to have the container element created. However, the container must exist when <code>init()</code> is run. <p>Required.</p> <ul style="list-style-type: none">• <code>params</code> – {Object} JSON object with viewer configuration parameters where the property name is either viewer-specific configuration option or SDK modifier, and the value of that property is a corresponding settings value. <p>Required.</p> <ul style="list-style-type: none">• <code>handlers</code> – {Object} JSON object with viewer event callbacks, where the property name is the name of supported viewer event and the property value is a JavaScript function reference to appropriate callback.
---------------	---

	<p>Optional.</p> <p>See Event callbacks for more information about viewer events.</p> <ul style="list-style-type: none">• <code>localizedTexts</code> – {Object} JSON object with localization data. Optional. <p>See Localization of user interface elements for more information.</p> <p>See also the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
--	---

Returns

None.

Example

```
var interactiveVideoViewer = new s7viewers.InteractiveVideoViewer({
  "containerId": "s7viewer",
  "params": {
    "asset": "/content/dam/mac/aodmarketingna/dm-viewers-content/video/Glacier.mp4",
    "config": "/etc/dam/presets/viewer/Shoppable_Video_Dark",
    "serverurl": "https://aodmarketingna.assetsadobe.com/is/image/",
    "videoseverurl": "https://gateway-na.assetsadobe.com/DMGateway/public/aodmarketingna",
    "contenturl": "https://aodmarketingna.assetsadobe.com/",
    "interactivedata": "is/content/content/dam/mac/aodmarketingna/_VIT/dm-viewers-content/video/Glacier.mp4.svideo.vtt"
  },
  "handlers": {
    "initComplete": function() {
      console.log("init complete");
    }
  },
  "localizedTexts": {
    "en": {
      "VideoPlayer.ERROR": "Your Browser does not support HTML5 Video tag or the video cannot be played."
    },
    "fr": {
      "VideoPlayer.ERROR": "Votre navigateur ne prend pas en charge la vidéo HTML5 tag ou la vidéo ne peuvent pas être lus."
    }
  },
  defaultLocale: "en"
});
```

setAsset

JavaScript API reference for Video Video Viewer.

```
setAsset(asset[, data])
```

Sets the new asset and optional additional asset data. You can call this parameter at any time, either before or after `init()`. If it is called after `init()`, the viewer swaps the asset at runtime.

See also [init](#).

asset	{String} new asset ID.
data	{JSON} JSON object with the following optional fields (case sensitive):

- `posterimage` – Image to display on the first frame before the video starts playing. See [VideoPlayer.posterimage](#).
- `caption` – location of the new caption file. If not specified, the caption button is not visible in the user interface.
- `navigation` – URL or path to the WebVTT navigation content. The WebVTT file should be served by Image Serving.
- `interactiveData` – URL or path to WebVTT interactive data content. The WebVTT file must be served by Image Serving.

Returns

None.

Example

```
<instance>.setAsset( "/content/dam/mac/aodmarketingna/dm-viewers-content/video/Glacier.mp4" )
```

setContainerId

JavaScript API reference for Video Video Viewer.

```
setContainerId(containerId)
```

Sets the ID of the DOM container (normally a DIV) into which the viewer is inserted. It is not necessary to have the container element created by the time this method is called. However, the container must exist when `init()` is run. It must be called before `init()`.

This method is optional if the viewer configuration information is passed with the `config` JSON object to the constructor.

Parameter

<i>containerId</i>	{string} ID of container.
--------------------	---------------------------

Returns

None.

Example

```
<instance>.setContainerId( "s7viewer" );
```

setHandlers

JavaScript API reference for Video Video Viewer

```
setHandlers(handlers)
```

Specifies zero or more callback handlers. A call to this method fully overwrites event handlers that were previously assigned for that viewer instance. Must be called before `init()`.

Parameter

<i>handlers</i>	<p>{Object} JSON object with viewer event callbacks, where the property name is the name of the supported viewer event and the property value is a JavaScript function reference to an appropriate callback.</p> <p>See Event callbacks for more information about viewer events.</p>
-----------------	---

Returns

None.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    console.log("init complete");
  }
})
```

setLocalizedTexts

JavaScript API reference for Video Video Viewer.

setLocalizedTexts(*localizationInfo*)

Sets localization SYMBOL values for one or more locales. This parameter must be called before `init()`.

<i>localizationInfo</i>	<p>{Object} JSON object with localization data.</p> <p>See Localization of user interface elements for more information.</p> <p>See also the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
-------------------------	--

See also [init](#).

Returns

None.

Example

```
<instance>.setLocalizedTexts({ "en":{"VideoPlayer.ERROR":"Your Browser does not support HTML5 Video tag or the video cannot be played."}, "fr":{"VideoPlayer.ERROR":"Votre navigateur ne prend pas en charge la vid   HTML5 tag ou la vid   ne peuvent pas   tre lus."}, defaultLocale:"en" })
```

setParam

JavaScript API reference for Video Video Viewer.

setParam(*name*, *value*)

Sets the viewer parameter to a specified value. The parameter is either a viewer-specific configuration option or a software development kit modifier. This parameter is called before `init()`.

This method is optional if the viewer configuration information was passed with `config` JSON object to constructor.

See also [init](#).

Parameters

<i>name</i>	{string} name of parameter.
<i>value</i>	{string} value of parameter. The value cannot be percent-encoded.

Returns

None.

Example

```
<instance>.setParam("style", "customStyle.css")
```

setParams

JavaScript API reference for Video Video Viewer.

```
setParams(params)
```

Sets one or more parameters to a given value. The method argument syntax is identical to a URL query string. That is, it represents name=value pairs separated with &. Just as in a query string, the names and values are percent-encoded using UTF8. Before you call `init()`, this parameter must be called.

This method is optional if the viewer configuration information was passed with `config` JSON object to constructor.

See also [init](#).

Parameter

<i>params</i>	{string} name=value parameter pairs separated with &.
---------------	---

Returns

None.

Example

```
<instance>.setParams("style=customStyle.css&VideoPlayer.playback=progressive")
```

Event callbacks

The viewer supports JavaScript event callbacks that the web page uses to track the viewer initialization process or runtime behavior.

Callback handlers are assigned by passing event names and corresponding handler functions with the `handlers` property to `config` JSON object in the viewer's constructor. Alternatively, it is possible to use `setHandlers()` API method.

Supported viewer events include the following:

- `initComplete` – triggers when viewer initialization is complete and all internal components are created, so that it is possible to use `getComponent()` API. The callback handler does not take any arguments.
- `trackEvent` – triggers each time an event occurs inside the viewer which may be handled by an event tracking system, such as Adobe Analytics. The callback handler takes the following arguments:
 - `objID` {String} not currently used.
 - `compClass` {String} not currently used.
 - `instName` {String} an instance name of the Viewer SDK component that triggered the event.
 - `timeStamp` {Number} event time stamp.
 - `eventInfo` {String} event payload.
- `quickViewActivate` – triggers when a user clicks or taps on interactive swatch within the interactive swatches component or in the "call to action" screen shown in the end of video playback. Callback handler takes the only argument which is a JSON object with the following fields:
 - `sku` {String} SKU value that is associated with the interactive swatch.
 - `<additionalVariable>` {String} zero or more additional variables associated with the interactive swatch.

See also [InteractiveVideoViewer](#) and [setHandlers](#).

Customizing Interactive Video Viewer

All visual customization and most behavior customization for the Interactive Video Viewer is done by creating a custom CSS.

The suggested workflow is to take the default CSS file for the appropriate viewer, copy it to a different location, customize it, and specify the location of the customized file in the `style=` command.

Default CSS files can be found at the following location:

```
<s7_viewers_root>/html5/InteractiveVideoViewer_dark.css
```

The viewer is supplied with two out-of-the-box CSS files, for "light" and "dark" color schemes. The "dark" version is used by default, but it is easy to switch to the "light" version by using the following standard CSS:

```
<s7_viewers_root>/html5/InteractiveVideoViewer_light.css
```

Custom CSS file must contain the same class declarations as the default one. If a class declaration is omitted, the viewer does not function properly because it does not provide built-in default styles for user interface elements.

An alternative way to provide custom CSS rules is to use embedded styles directly on the web page or in one of linked external CSS rules.

When creating custom CSS keep in mind that the viewer assigns `.s7interactivevideoviewer` class to its container DOM element. If you are using external CSS file passed with `style=` command, use `.s7interactivevideoviewer` class as parent class in descendant selector for your CSS rules. If you are doing embedded styles on the web page, additionally qualify this selector with an ID of the container DOM element as follows:

```
#<containerId>.s7interactivevideoviewer
```

Building responsive designed CSS

It is possible to target different devices and embedding sizes in CSS to make your content display differently depending on a user's device or a particular web page layout. This includes, but is not limited to, different layouts, user interface element sizes, and artwork resolution.

The viewer supports two mechanisms of creating responsive designed CSS: CSS markers and standard CSS media queries. You can use these independently or together.

CSS markers

To assist in creating responsive designed CSS, the viewer supports CSS markers. These are special CSS classes that are dynamically assigned to the top-level viewer container element based on the run-time viewer size and the input type used on the current device.

The first group of CSS markers includes `.s7size_large`, `.s7size_medium`, and `.s7size_small` classes. They are applied based on the run-time area of the viewer container. If the viewer area is equal or bigger than the size of a common desktop monitor then `.s7size_large` is used; if the area is close to a common tablet device then `.s7size_medium` is assigned. For areas similar to mobile phone screens then `.s7size_small` is set. The primary purpose of these CSS markers is to create different user interface layouts for different screens and viewer sizes.

The second group of CSS Markers contains `.s7mouseinput` and `.s7touchinput`. `.s7touchinput` is set if the current device has touch input capabilities; otherwise, `.s7mouseinput` is used. These markers are mostly intended to create user interface input elements with different screen sizes for different input types, because normally touch input requires larger elements.

The third group of CSS Markers contains `.s7device_landscape` and `.s7device_portrait`. `.s7device_landscape` is set if the touch device is in landscape orientation; `.s7device_portrait` is used when the touch device is rotated to portrait orientation. These CSS markers are intended for use on desktop systems only.

The following sample CSS sets the play/pause button size to 28x28 pixels on systems with mouse input and 56x56 pixels on touch devices. In addition, it hides the button completely if the viewer size is reduced significantly:

```
.s7interactivevideoviewer.s7mouseinput .s7playpausebutton {
  width:28px;
  height:28px;
}
.s7interactivevideoviewer.s7touchinput .s7playpausebutton {
  width:56px;
  height:56px;
}
.s7interactivevideoviewer.s7size_small .s7playpausebutton {
  visibility:hidden;
}
```

In this next example, the video control bar is position 138 pixels above the bottom of the viewer if the touch device is in portrait orientation, and move it to the very bottom of the viewer in all other cases:

```
.s7interactivevideoviewer.s7touchinput.s7device_landscape .s7controlbar,
.s7interactivevideoviewer.s7mouseinput .s7controlbar {
  bottom:0px;
}
.s7interactivevideoviewer.s7touchinput.s7device_portrait .s7controlbar {
  bottom:136px;
}
```

To target devices with different pixel density you need to use CSS media queries. The following media query block would contain CSS specific to high-density screens:

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5)
{
}
```

Using CSS markers is the most flexible way of building responsive designed CSS because it lets you target not only the device screen size but the actual viewer size, which is useful for responsive design layouts.

You can use the default viewer CSS file as an example of a CSS Markers approach.

CSS media queries

You can also accomplish device sensing by using pure CSS media queries. Everything enclosed within a given media query block is applied only when it is run on a corresponding device.

When applied to Mobile Viewers use four CSS media queries, defined in your CSS, in the order listed below:

1. Contains only rules specific for all touch devices.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and (max-device-width:799px),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in)
and (max-device-height:799px)
{
}
```

2. Contains only rules specific for tablets with high resolution screens.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px) and (-webkit-min-device-pixel-ratio:1.5),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px) and (-webkit-min-device-pixel-ratio:1.5)
{
}
```

3. Contains only rules specific for all mobile phones.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in)
{
}
```

4. Contains only rules specific for mobile phones with high resolution screens.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in) and
(-webkit-min-device-pixel-ratio: 1.5),
only screen and (device-width:720px) and (device-height:1280px) and
(-webkit-device-pixel-ratio: 2),
only screen and (device-width:1280px) and (device-height:720px) and
(-webkit-device-pixel-ratio: 2)
{
}
```

Using a media queries approach, you should organize CSS with device sensing as follows:

- First, the desktop-specific section defines all properties that are either desktop-specific or common to all screens.
- And second, the four media queries go in the order defined above and provide CSS rules that are specific for the corresponding device type.

There is no need to duplicate the entire viewer CSS in each media query. Only properties that are specific to given devices are redefined inside a media query.

CSS Sprites

Many viewer user interface elements are styled using bitmap artwork and have more than one distinct visual state. A good example is a button that normally has at least 3 different states: "up", "over", and "down". Each state requires its own bitmap artwork assigned.

With a classic approach to styling, the CSS would have a separate reference to individual image file on the server for each state of the user interface element. The following is a sample CSS for styling a full-screen button:

```
.s7interactivevideoviewer.s7mouseinput .s7playpausebutton[selected='true'][state='up'] {
background-image:url(images/v2/PlayButton_up.png);
}
.s7interactivevideoviewer.s7mouseinput .s7playpausebutton[selected='true'][state='over'] {
background-image:url(images/v2/PlayButton_over.png);
}
.s7interactivevideoviewer.s7mouseinput .s7playpausebutton[selected='true'][state='down'] {
background-image:url(images/v2/PlayButton_down.png);
}
.s7interactivevideoviewer.s7mouseinput .s7playpausebutton[selected='true'][state='disabled'] {
background-image:url(images/v2/PlayButton_disabled.png);
}
.s7interactivevideoviewer.s7mouseinput .s7playpausebutton[selected='false'][state='up'] {
background-image:url(images/v2/PauseButton_up.png);
}
.s7interactivevideoviewer.s7mouseinput .s7playpausebutton[selected='false'][state='over'] {
background-image:url(images/v2/PauseButton_over.png);
}
.s7interactivevideoviewer.s7mouseinput .s7playpausebutton[selected='false'][state='down'] {
background-image:url(images/v2/PauseButton_down.png);
}
.s7interactivevideoviewer.s7mouseinput .s7playpausebutton[selected='false'][state='disabled'] {
background-image:url(images/v2/PauseButton_disabled.png);
}
.s7interactivevideoviewer.s7mouseinput
.s7playpausebutton[selected='true'][replay='true'][state='up'] {
background-image:url(images/v2/ReplayButton_up.png);
}
.s7interactivevideoviewer.s7mouseinput
.s7playpausebutton[selected='true'][replay='true'][state='over'] {
background-image:url(images/v2/ReplayButton_over.png);
}
.s7interactivevideoviewer.s7mouseinput
.s7playpausebutton[selected='true'][replay='true'][state='down'] {
background-image:url(images/v2/ReplayButton_down.png);
}
.s7interactivevideoviewer.s7mouseinput
.s7playpausebutton[selected='true'][replay='true'][state='disabled'] {
background-image:url(images/v2/ReplayButton_disabled.png);
}
```

The drawback to this approach is that the end user experiences flickering or delayed user interface response when the element is interacted with for the first time. This action occurs because the image artwork for the new element state is not yet downloaded. Also, this approach may have a slight negative impact on performance because of an increase in the number of HTTP calls to the server.

CSS sprites is a different approach where image artwork for all element states is combined into a single PNG file called a "sprite". Such "sprite" has all visual states for the given element positioned one after another. When styling a user interface element with sprites the same sprite image is referenced for all different states in the CSS. Also, the background-position property is used for each state to specify which part of the "sprite" image is used. You can structure a "sprite" image in any suitable way. Viewers normally have it vertically stacked. Below is a "sprite"-based example of styling the same full-screen button earlier:

```
.s7interactivevideoviewer .s7fullscreenbutton[state][selected]{
background-image: url(images/v2/FullScreenButton_dark_sprite.png);
}
.s7interactivevideoviewer.s7mouseinput .s7fullscreenbutton[selected='true'][state='up'] {
background-position: -84px -1148px;
}
.s7interactivevideoviewer.s7mouseinput .s7fullscreenbutton[selected='true'][state='over'] {
background-position: -56px -1148px;
}
```

```
.s7interactivevideoviewer.s7mouseinput .s7fullscreenbutton[selected='true'][state='down'] {
background-position: -28px -1148px;
}
.s7interactivevideoviewer.s7mouseinput .s7fullscreenbutton[selected='true'][state='disabled']
{
background-position: -0px -1148px;
}
.s7interactivevideoviewer.s7mouseinput .s7fullscreenbutton[selected='false'][state='up'] {
background-position: -84px -1120px;
}
.s7interactivevideoviewer.s7mouseinput .s7fullscreenbutton[selected='false'][state='over'] {
background-position: -56px -1120px;
}
.s7interactivevideoviewer.s7mouseinput .s7fullscreenbutton[selected='false'][state='down'] {
background-position: -28px -1120px;
}
.s7interactivevideoviewer.s7mouseinput .s7fullscreenbutton[selected='false'][state='disabled']
{
background-position: -0px -1120px;
}
```

General styling notes and advice

- When customizing the viewer user interface with CSS the use of the `!IMPORTANT` rule is not supported to style viewer elements. In particular, `!IMPORTANT` rule should not be used to override any default or run-time styling provided by the viewer or Viewer SDK. The reason is that it may affect the behavior of proper components. Instead, you should use CSS selectors with the proper specificity to set CSS properties that are documented in this reference guide.
- All paths to external assets within CSS are resolved against the CSS location, not the viewer HTML page location. Be aware of this rule when you copy the default CSS to a different location. Either copy the default assets as well or update paths within the custom CSS.
- The preferred format for bitmap artwork is PNG.
- Bitmap artwork is assigned to user interface elements using the `background-image` property.
- The width and height properties of a user interface element define its logical size. The size of the bitmap passed to `background-image` does not affect logical size.
- To use the high pixel density of high-resolution screens like Retina, specify bitmap artwork twice as large as the logical user interface element size. Then, apply the `-webkit-background-size:contain` property to scale the background down to the logical user interface element size.
- To remove a button from the user interface, add `display:none` to its CSS class.
- You can use various formats for color value that CSS supports. If you need transparency, use the format `rgba(R,G,B,A)`. Otherwise, you can use the format `#RRGGBB`.

Common User Interface Elements

The following is user interface elements reference documentation that applies to Video Video Viewer:

Call to action

The Call to action panel appears when the video ends and displays all interactive swatches associated with the particular video.

The panel consists of a header area showing the video title, a replay button in the upper-right corner, and actual interactive swatches shown as a scrollable grid. You can disable the panel using the [callToActionRecap](#) configuration attribute.

The call to action panel always takes the entire available viewer area.

The following CSS class selector controls the appearance of the background color in the call to action panel:

```
.s7interactivevideoviewer .s7calltoaction
```

CSS properties of the background color of the call to action panel

background-color	Background color of the call to action panel.
------------------	---

Example

To set up a call to action panel with dark gray background:

```
.s7interactivevideoviewer .s7calltoaction {
  background-color: #222222;
}
```

The following CSS class selector controls the appearance of the header in the call to action panel:

```
.s7interactivevideoviewer .s7calltoaction .s7header
```

CSS properties of the call to action panel header

background-color	Background color of the header.
height	Height of the header.
border-bottom	Bottom border of the header.

Example

To set up a header that is 70 pixels tall, with a dark gray background, and a slightly lighter gray two pixel border along the bottom:

```
.s7interactivevideoviewer .s7calltoaction .s7header {
  height: 70px;
  border-bottom: 2px solid #444444;
  background-color: #222222;
}
```

The following CSS class selector controls the appearance of the header title in the call to action panel:

```
.s7interactivevideoviewer .s7calltoaction .s7header .s7title
```

CSS properties of the header title in the call to action panel:

color	Text color inside the banner.
font-size	Font size.
line-height	Line height.
font-family	Font family.
text-align	Alignment of text in the banner.
padding-left	Left padding.
padding-right	Right padding to allow space for the Replay button.

Example

To set up a video title with a 70 pixel line height, 25 pixel font size, white color, and left aligned:

```
.s7interactivevideoviewer .s7calltoaction .s7header .s7title {
  line-height: 70px;
  font-size: 25px;
  color: #ffffff;
  text-align: left;
}
```

The following CSS class selector controls the appearance of the close button in the call to action panel:

```
.s7interactivevideoviewer .s7calltoaction .s7closebutton
```

CSS properties of the close button in the call to action panel:

top	Position from the top of the header, including padding.
right	Position from the right of the header, including padding.
width	Button width.
height	Button height.
background-image	Image displayed for a given button state.
background-position	Position inside the artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button support the *state* attribute selector, which can be used to apply different skins to different button states.

Example

To set up a replay button that is 28 x 28 pixels; positioned 20 pixels from the top and from the right edge of the header; displays a different image for each of the four different button states; takes the artwork from the component's sprite image:

```
.s7interactivevideoviewer .s7calltoaction .s7closebutton {
  top: 20px;
  right: 20px;
  background-size: 336px;
  width: 28px;
  height: 28px;
}
.s7interactivevideoviewer .s7calltoaction .s7closebutton[state] {
  background-image: url(images/v2/PlayPauseButton_sprite.png);
}
.s7interactivevideoviewer .s7calltoaction .s7closebutton[state='up'] {
  background-position: -28px -1260px;
}
.s7interactivevideoviewer .s7calltoaction .s7closebutton[state='over'] {
  background-position: -0px -1260px;
}
.s7interactivevideoviewer .s7calltoaction .s7closebutton[state='down'] {
  background-position: -28px -1232px;
}
.s7interactivevideoviewer .s7calltoaction .s7closebutton[state='disabled'] {
```

```
background-position: -0px -1232px;
}
```

The following CSS class selector controls the appearance of the thumbnail grid view in the call to action panel:

```
.s7interactivevideoviewer .s7calltoaction .s7thumbnailgridview
```

CSS properties of the thumbnail grid view in the call to action panel:

background-color	Background color of the thumbnails area.
------------------	--

Example

To set up a thumbnails area with a dark gray background:

```
.s7interactivevideoviewer .s7calltoaction .s7thumbnailgridview {
  background-color: #222222;
}
```

The following CSS class selector controls the appearance of the thumb cell in the call to action panel:

```
.s7interactivevideoviewer .s7calltoaction .s7thumbcell
```

CSS properties of the thumbcell in the call to action panel:

margin	<p>Size of the horizontal and vertical margin around each thumbnail.</p> <p>Actual horizontal thumbnail spacing equals the sum of the left and right margin set for <code>.s7thumbcell</code>. The same rule also applies but to vertical spacing.</p>
--------	--

Example

To set up horizontal spacing of 24 pixels and vertical spacing of 18 pixels:

```
.s7interactivevideoviewer .s7calltoaction .s7thumbcell {
  margin-top: 9px;
  margin-bottom: 9px;
  margin-left: 12px;
  margin-right: 12px;
}
```

The following CSS class selector controls the appearance of the thumbnail in the call to action panel:

```
.s7interactivevideoviewer .s7calltoaction .s7thumb
```

CSS properties of the thumbnail in the call to action panel:

width	Width of the thumbnail.
height	Height of the thumbnail.
border	Border of the thumbnail.



Note: Thumbnail supports the `state` attribute selector, which can be used to apply different skins to different thumbnail states. In particular, `state="selected"` corresponds to the thumbnail for the image currently selected; `state="default"` corresponds to the rest of thumbnails; `state="over"` is used on mouse hover.

Example

To set up thumbnails that are 94 x 100 pixels:

```
.s7interactivevideoviewer .s7calltoaction .s7thumb {
  width:94px;
  height:100px;
}
```

The following CSS class selector controls the appearance of the thumbnail label in the call to action panel:

```
.s7interactivevideoviewer .s7calltoaction .s7label
```

CSS properties of the thumbnail label in the call to action panel:

color	Text color of label.
text-align	Horizontal alignment of label.
font-family	Font name.
font-size	Font family.

Example

To set up labels that use a white color, be center-aligned 15 pixels, and use an Arial font:

```
.s7interactivevideoviewer .s7calltoaction .s7label {
  text-align: center;
  color: #ffffff;
  font-family: Arial,Helvetica,sans-serif;
  font-size: 15px;
}
```

If there are more thumbnails than can fit vertically into view, thumbnails renders a vertical scroll bar on the right side. By default, the call to action panel renders a tiny vertical bar without thumb and scroll buttons. However, it is possible to customize the bar by altering the viewer CSS.

The following CSS class selector controls the appearance of the scroll bar area in the call to action panel:

```
.s7interactivevideoviewer .s7calltoaction .s7thumbnailgridview .s7scrollbar
```

CSS properties of the scroll bar area in the call to action panel:

width	Width of scroll bar.
top	Vertical scroll bar offset from the top of the thumbnails area.
bottom	Vertical scroll bar offset from the bottom of the thumbnails area.
right	Horizontal scroll bar offset from the right edge of the thumbnails area.

Example

To set up a scroll bar that is 22 pixels wide and does not have any margin from the top, right, or bottom of the thumbnails area:

```
.s7interactivevideoviewer .s7calltoaction .s7thumbnailgridview .s7scrollbar {
  top:0px;
  bottom:0px;
  right:0px;
  width:22px;
}
```

The scroll bar track is the area between the top and bottom scroll bar buttons. The component automatically sets the position and height of the track.

The following CSS class selector controls the appearance of the scroll bar track in the call to action panel:

```
.s7interactivevideoviewer .s7calltoaction .s7thumbnailgridview .s7scrollbar .s7scrolltrack
```

CSS properties of the scroll track bar

width	Width of scroll track bar.
background-color	Background color of the track bar.

Example

To set up a scroll bar track that is 22 pixels wide and has a gray color:

```
.s7interactivevideoviewer .s7calltoaction .s7thumbnailgridview .s7scrollbar .s7scrolltrack {
  width:22px;
  background-color:#222222;
}
```

The scroll bar thumb moves vertically within the scroll track area. Its vertical position is fully controlled by the component logic; however, the thumb height does not dynamically change depending on the amount of content.

The following CSS class selector controls the appearance of the thumb height and other aspect:

```
.s7interactivevideoviewer .s7calltoaction .s7thumbnailgridview .s7scrollbar .s7scrollthumb
```

CSS properties of the thumb height in the call to action panel:

width	Width of thumb.
height	Height of thumb.
padding-top	Vertical padding between the top of the track.
padding-bottom	Vertical padding between the bottom of the track.
border-radius	Radius of border.
background-color	Color of thumb.
background-image	Image displayed for a given thumb state.

background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
---------------------	---



Note: Thumb supports the `state` attribute selector, which can be used to apply different skins to the following different thumb states: "up", "down", "over", and "disabled".

Example

To set up a scroll bar thumb that is 6 x 167 pixels, has three pixel rounded corners, and a gray color:

```
.s7interactivevideoviewer .s7calltoaction .s7thumbnailgridview .s7scrollbar .s7scrollthumb[state]
{
  width:6px;
  height:167px;
  background-color:#666666;
  background-image:none;
  border-radius:3px 3px 3px 3px;
}
```

The following CSS class selector controls the appearance of the top and bottom scroll buttons:

```
.s7interactivevideoviewer .s7calltoaction .s7thumbnailgridview .s7scrollbar .s7scrollupbutton
.s7interactivevideoviewer .s7calltoaction .s7thumbnailgridview .s7scrollbar .s7scrolldownbutton
```

It is not possible to position scroll buttons using CSS top, left, bottom, or right properties; the viewer logic positions them automatically. The call to action panel in the interactive video viewer does not use these buttons in the scroll bar, so their size is set to 0 pixels in the default CSS.

CSS properties of the top and bottom scroll buttons in the call to action panel:

width	Width of button.
height	Height of button.
background-image	Image displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: These buttons support the `state` attribute selector, which can be used to apply different skins to the following different thumb states: "up", "down", "over", and "disabled".

The button tool tips can be localized. See [Localization of user interface elements](#).

Example

Disable scroll buttons by setting their size to 0 and hiding them:

```
.s7interactivevideoviewer .s7calltoaction .s7thumbnailgridview .s7scrollbar .s7scrollupbutton
{
  visibility: hidden;
  width: 0px;
```

```
height: 0px;
}
.s7interactivevideoviewer .s7calltoaction .s7thumbnailgridview .s7scrollbar .s7scrollbutton
{
visibility: hidden;
width: 0px;
height: 0px;
}
```

Caption button

This button toggles closed caption display on and off. It is not visible if the caption parameter is not specified.


You can size, skin, and position this button, relative to the control bar that contains it, by using CSS.

The appearance of this button is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7closedcaptionbutton
```

CSS properties of the caption button

top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	The width of the full screen button.
height	The height of the full screen button.
background-image	The displayed image for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

 **Note:** This button supports both the *state* and *selected* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the state when captions are visible and *selected='false'* is used when captions are hidden.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example

To set up a closed caption button that is 28 x 28 pixels, positioned four pixels from the top and 68 pixels from the right edge of the control bar, and displays a different image for each of the four different button states when selected or not selected.

```
.s7interactivevideoviewer .s7closedcaptionbutton {
position: absolute;
top: 4px;
right: 68px;
```

```
width:28px;
height:28px;
}
.s7interactivevideoviewer .s7closedcaptionbutton[selected='true'][state='up'] {
background-image:url(images/v2/ClosedCaptionButton_up.png);
}
.s7interactivevideoviewer .s7closedcaptionbutton[selected='true'][state='over'] {
background-image:url(images/v2/ClosedCaptionButton_over.png);
}
.s7interactivevideoviewer .s7closedcaptionbutton[selected='true'][state='down'] {
background-image:url(images/v2/ClosedCaptionButton_down.png);
}
.s7interactivevideoviewer .s7closedcaptionbutton[selected='true'][state='disabled'] {
background-image:url(images/v2/ClosedCaptionButton_disabled.png);
}
.s7interactivevideoviewer .s7closedcaptionbutton[selected='false'][state='up'] {
background-image:url(images/v2/ClosedCaptionButton_disabled.png);
}
.s7interactivevideoviewer .s7closedcaptionbutton[selected='false'][state='over'] {
background-image:url(images/v2/ClosedCaptionButton_over.png);
}
.s7interactivevideoviewer .s7closedcaptionbutton[selected='false'][state='down'] {
background-image:url(images/v2/ClosedCaptionButton_down.png);
}
.s7interactivevideoviewer .s7closedcaptionbutton[selected='false'][state='disabled'] {
background-image:url(images/v2/ClosedCaptionButton_disabled.png);
}
```

Control bar

The control bar is the rectangular area that contains and sits behind all the UI controls available for the video viewer, such as the play/pause button, volume controls, and so on.

The control bar always takes the entire available viewer width. It is possible to change its color, height, and vertical position by CSS, relative to the video viewer container.

The following CSS class selector controls the appearance of the control bar:

```
.s7interactivevideoviewer .s7controlbar
```

CSS properties of the control bar

top	Position from the top border, including padding.
bottom	Position from the bottom border, including padding.
height	Height of the control bar.
background-color	Background color of the control bar.

Example

To set up a video viewer with a gray control bar that is 30 pixels tall and sits at the top of the video viewer container.

```
.s7interactivevideoviewer .s7controlbar {
position: absolute;
top: 0px;
height: 30px;
background-color: rgb(51, 51, 51);
}
```

Facebook share

Facebook share tool consists of a button added to the Social share panel. When the button is clicked the user is redirected to a sharing dialog box that is provided by a social service. The position of the button is fully managed by the Social share tool.

The appearance of the Facebook share button is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7facebookshare
```

CSS properties of the Facebook share tool

width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#).

Example

To set up a Facebook share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```
.s7interactivevideoviewer .s7facebookshare {
  width:28px;
  height:28px;
}
.s7interactivevideoviewer .s7facebookshare[state='up'] {
background-image:url(images/v2/FacebookShare_dark_up.png);
}
.s7interactivevideoviewer .s7facebookshare[state='over'] {
background-image:url(images/v2/FacebookShare_dark_over.png);
}
.s7interactivevideoviewer .s7facebookshare[state='down'] {
background-image:url(images/v2/FacebookShare_dark_down.png);
}
.s7interactivevideoviewer .s7facebookshare[state='disabled'] {
background-image:url(images/v2/FacebookShare_dark_disabled.png);
}
```

Focus highlight

Input focus highlight displayed around focused viewer user interface element is controlled with the CSS class selector.

CSS properties

The appearance is controlled with the following CSS class selector:

```
.s7interactivevideoviewer *:focus
```


CSS property	Description
outline	Focus highlight style.

Example – to disable the default browser focus highlight for all viewer user interface elements, add the following CSS selector to viewer's style sheet:

```
.s7interactivevideoviewer *:focus {
  outline: none;
}
```

Full screen button

The full screen button causes the video player to enter or exit full screen mode when a user clicks it.

You can size, skin, and position the full screen button, relative to the control bar that contains it, by CSS.

The appearance of the full screen button is controlled with the CSS class selector:

```
.s7interactivevideoviewer .s7fullscreenbutton
```

CSS properties of the full screen button

top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	The width of the full screen button.
height	The height of the full screen button.
background-image	The displayed image for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports both the `state` and `selected` attribute selectors, which can be used to apply different skins to different button states. In particular, `selected='true'` corresponds to the "full screen" state and `selected='false'` corresponds to the "normal" state.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example

To set up a full screen button that is 32 x 32 pixels, and positioned 6 pixels from the top and right edge of the control bar. Also, display a different image for each of the four different button states when selected or not selected.

```
.s7interactivevideoviewer .s7fullscreenbutton {
top:6px;
right:6px;
width:32px;
height:32px;
}
.s7interactivevideoviewer .s7fullscreenbutton [selected='false'][state='up'] {
background-image:url(images/enterFullBtn_up.png);
}
.s7interactivevideoviewer .s7fullscreenbutton [selected='false'][state='over'] {
background-image:url(images/enterFullBtn_over.png);
}
.s7interactivevideoviewer .s7fullscreenbutton [selected='false'][state='down'] {
background-image:url(images/enterFullBtn_down.png);
}
.s7interactivevideoviewer .s7fullscreenbutton [selected='false'][state='disabled'] {
background-image:url(images/enterFullBtn_disabled.png);
}
.s7interactivevideoviewer .s7fullscreenbutton [selected='true'][state='up'] {
background-image:url(images/exitFullBtn_up.png);
}
.s7interactivevideoviewer .s7fullscreenbutton [selected='true'][state='over'] {
background-image:url(images/exitFullBtn_over.png);
}
.s7interactivevideoviewer .s7fullscreenbutton [selected='true'][state='down'] {
background-image:url(images/exitFullBtn_down.png);
}
.s7interactivevideoviewer .s7fullscreenbutton [selected='true'][state='disabled'] {
background-image:url(images/exitFullBtn_disabled.png);
}
```

Icon effect

The play icon is overlaid on the main view area. It displays when the video is paused, or when the end of the video is reached, and it also depends on the `iconeffect` parameter.

The appearance of the play icon is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7videoplayer .s7iconeffect
```

CSS properties of the play icon

background-image	The displayed image for the play icon.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
width	The width of the play icon.
height	The height of the play icon.

Icon effect supports the state attribute selector. `state="play"` is used when the video is paused in the middle of playback, and `state="replay"` is used when the play head is in the end of the stream.

Example

Setup a 100 x 100 pixel play icon.

```
.s7interactivevideoviewer .s7videoplayer .s7iconeffect {
  width: 100px;
  height: 100px;}
.s7interactivevideoviewer .s7videoplayer .s7iconeffect[state="play"] {
background-image: url(images/playIcon.png);
}
.s7interactivevideoviewer .s7videoplayer .s7iconeffect[state="replay"] {
background-image: url(images/replayIcon.png);
}
```

Interactive swatches

The interactive swatches panel appears next to the video content if interactive data was passed to the viewer in configuration. It consists of a banner at the top that renders text such as "Click to View", a column of one or more interactive swatches and two scroll buttons (available only on desktop systems).

On desktop systems and on touch devices, in landscape orientation, interactive swatches are rendered vertically to the right of the video content. On touch devices in portrait orientation they appear at the bottom of the viewer, as a horizontal row of swatches.

The following CSS class selector controls the location and orientation of the interactive swatches panel:

```
.s7interactivevideoviewer .s7interactivесwatches
```

CSS properties of the interactive swatches

width	Width of the interactive swatches panel
height	Height of the interactive swatches panel.
top	Top position of he interactive swatches panel.
bottom	Bottom position of the interactive swatches panel.
left	Left position of the interactive swatches panel.
right	Right position of the interactive swatches panel.

The run-time location and orientation of the interactive swatches panel is defined by a combination of the above CSS properties as follows:

- To render interactive swatches horizontally at the bottom of the viewer, set the height to an absolute pixel value; left and bottom to 0px; width, right, and top to auto.
- To render interactive swatches vertically to the right of the video content, set the width to an absolute pixel; right and top to 0px; height, left and bottom to auto.

It is possible to use CSS markers in conjunction with this styling to achieve adaptive placement of the interactive swatches panel.

Example

To set up an interactive swatches panel to render horizontally at the bottom of the viewer on touch devices in landscape orientation and to show vertically to the right of the video content in all other cases:

```
.s7interactivevideoviewer.s7touchinput.s7device_landscape .s7interactiveswatches,
.s7interactivevideoviewer.s7mouseinput .s7interactiveswatches {
  width:120px;
  height:auto;
  right:0px;
  top:0px;
  left:auto;
  bottom:auto;
}
.s7interactivevideoviewer.s7touchinput.s7device_portrait .s7interactiveswatches {
  width:auto;
  height:136px;
  right:auto;
  top:auto;
  left:0px;
  bottom:0px;
}
```

The size and position of the banner is managed by the interactive swatches component based on other styling applied with CSS and cannot be set explicitly.

The following CSS class selector controls the appearance of the banner panel:

```
.s7interactivevideoviewer .s7interactiveswatches .s7banner
```

CSS properties of the banner panel

background-color	Background color of the banner panel.
color	Text color inside the banner panel.
border	Border around the banner panel.
font-weight	The font weight to use for the text inside the banner panel.
font-size	The font size to use for the text inside the banner panel.
font-family	The font family to use for the text inside the banner panel.
font-align	The font alignment to use for the text inside the banner panel.

The banner tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example

To set up a banner with dark gray background, lighter gray two pixel border and the white text centered horizontally:

```
.s7interactivevideoviewer .s7interactiveswatches .s7banner {
  background-color: #222222;
  border-bottom: 2px solid #444444;
  color: #ffffff;
  text-align: center;
}
```

The following CSS class selector controls the appearance of the swatches:

```
.s7interactivevideoviewer .s7interactiveswatches .s7swatches
```

CSS properties of the swatches area

background-color	Background color of the swatches area.
------------------	--

Example

To set up swatches area with dark gray background:

```
.s7interactivevideoviewer .s7interactiveswatches .s7swatches {  
    background-color: #222222;  
}
```

The following CSS class selector controls the spacing between swatch thumbnails:

```
.s7interactivevideoviewer .s7interactiveswatches .s7swatches .s7thumbcell
```

CSS properties of the swatches thumbnail spacing

margin	The size of the horizontal and vertical margin around each thumbnail. The actual thumbnail spacing equals to the sum of the left and right margin set for .s7thumbcell.
--------	---

Example

To set up vertical spacing to be 10 pixels:


```
.s7interactivevideoviewer .s7interactiveswatches .s7swatches .s7thumbcell {  
    margin: 5px;  
}
```

The following CSS class selector controls the appearance of individual thumbnails:

```
.s7interactivevideoviewer .s7interactiveswatches .s7swatches .s7thumb
```

CSS properties of the appearance of individual thumbnails

width	Width of the thumbnail.
height	Height of the thumbnail.
border	Border of the thumbnail.

 **Note:** Thumbnail supports the *state* attribute selector, which you can use to apply different skins to different thumbnail states. In particular, *state="selected"* corresponds to the thumbnail for the currently selected image; *state="default"* corresponds to the rest of thumbnails; *state="over"* is used on mouse hover.

Example

To set up thumbnails that are 100 x 75 pixels:

```
.s7interactivevideoviewer .s7interactivewatches .s7swatches .s7thumb {
  width:100px;
  height:75px;
}
```

The following CSS class selector controls the appearance of the thumbnail label:

```
.s7interactivevideoviewer .s7interactivewatches .s7swatches .s7label
```

CSS properties of the appearance of the thumbnail label

color	Text color.
border	Label border.
text-align	Horizontal text alignment.
font-family	Font name.

Example

To set up labels to use left-aligned, white, 12 pixels, in Helvetica font, and a bottom border:

```
.s7interactivevideoviewer .s7interactivewatches .s7swatches .s7label {
  border-bottom: 1px solid #e9e9e9;
  text-align: left;
  color: #ffffff;
  font-family: Helvetica,sans-serif;
  font-size: 12px;
}
```


The following CSS class selector controls the appearance of the up and down scroll buttons:

```
.s7interactivevideoviewer .s7interactivewatches .s7scrollupbutton
.s7interactivevideoviewer .s7interactivewatches .s7scrolldownbutton
```

It is not possible to position the scroll buttons using CSS `top`, `left`, `bottom`, and `right` properties; instead, the viewer logic positions them automatically.

CSS properties of the appearance of the up and down scroll buttons

width	Width of the scroll button.
height	Height of the scroll button.
background-image	The image that is displayed for a given button state.
background-position	The position inside the artwork sprite, if CSS sprites are used. See also CSS Sprites .

 **Note:** This button supports the `state` attribute selector, which you can use to apply different skins to the button states: "up", "down", "over", and "disabled".

The button tool tips can be localized. See [Localization of user interface elements](#) for more information.

Example

To set up scroll up button that is 60 x 36 pixels, have different artwork for each state and takes that artwork from the component's sprite image:

```
.s7interactivevideoviewer .s7interactiveswatches .s7scrollupbutton {
  background-size: 120px;
  width: 60px;
  height: 36px;
}
.s7interactivevideoviewer .s7interactiveswatches .s7scrollupbutton[state] {
  background-image: url(images/v2/InteractiveSwatches_light_sprite.png);
}
.s7interactivevideoviewer .s7interactiveswatches .s7scrollupbutton[state='up'] {
  background-position: -60px -684px; }
.s7interactivevideoviewer .s7interactiveswatches .s7scrollupbutton[state='over'] {
  background-position: -0px -684px; }
.s7interactivevideoviewer .s7interactiveswatches .s7scrollupbutton[state='down'] {
  background-position: -60px -648px; }
.s7interactivevideoviewer .s7interactiveswatches .s7scrollupbutton[state='disabled'] {
  background-position: -0px -648px; }
```

Link share


Link share tool consists of a button added to the Social share panel and the modal dialog box that displays when the tool is activated. The position of the button is fully managed by the Social share tool.

The appearance of the link share button is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkshare
```

CSS properties of the link share tool

width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

 **Note:** This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#).

Example – to set up a link share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```
.s7interactivevideoviewer .s7linkshare {
  width:28px;
  height:28px;
}
.s7interactivevideoviewer .s7linkshare[state='up'] {
background-image:url(images/v2/LinkShare_dark_up.png);
}
.s7interactivevideoviewer .s7linkshare[state='over'] {
background-image:url(images/v2/LinkShare_dark_over.png);
}
.s7interactivevideoviewer .s7linkshare[state='down'] {
background-image:url(images/v2/LinkShare_dark_down.png);
}
.s7interactivevideoviewer .s7linkshare[state='disabled'] {
background-image:url(images/v2/LinkShare_dark_disabled.png);
}
```

The background overlay that covers the web page when the dialog box is active is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7backoverlay
```

CSS properties of the background overlay

opacity	Background overlay opacity.
background-color	Background overlay color.

Example – to set up a background overlay to be gray with 70% opacity:

```
.s7interactivevideoviewer .s7linkdialog .s7backoverlay {
  opacity:0.7;
  background-color:#222222;
}
```

By default the modal dialog box is displayed centered in the screen on desktop systems and takes the entire web page area on touch devices. In all cases, the positioning and sizing of the dialog box is managed by the component. The dialog box is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7dialog
```

CSS properties of the dialog box

border-radius	Dialog box border radius, in case the dialog box does not take the entire browser.
background-color	Dialog box background color.
width	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).
height	Should be either unset, or set to 100%, in which case the dialog box takes the entire browser window (this mode is preferred on touch devices).

Example – to set up the dialog box to use the entire browser window and have a white background on touch devices:

```
.s7interactivevideoviewer.s7touchinput .s7linkdialog .s7dialog {
  width:100%;
```



```
height:100%;
background-color: #ffffff;
}
```

The dialog box header consists of an icon, a title text, and a close button. The header container is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7dialogheader
```

CSS properties of the dialog box header

padding	Inner padding for header content.
---------	-----------------------------------

The icon and the title text are wrapped into an additional container controlled with the following CSS class selector:

```
.s7videoviewer .s7linkdialog .s7dialogheader .s7dialogline
```

CSS properties of the dialog line

padding	Inner padding for the header icon and title
---------	---

Header icon is controlled with the following CSS class selector

```
.s7interactivevideoviewer .s7linkdialog .s7dialogheadericon
```

CSS properties of the dialog box header icon

width	Icon width.
height	Icon height.
background-image	Icon image.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

Header title is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7dialogheadertext
```

CSS properties of the dialog box header text

font-weight	Font weight.
font-size	Font height.
font-family	Font family.
padding	Internal text padding.

Close button is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7closebutton
```

CSS properties of the close button

top	Vertical button position relative to header container.
right	Horizontal button position relative to header container.
width	Button width.
height	Button height.
padding	Inner padding of button.
background-image	Button image for each state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The Close button tool tip and the dialog box title can be localized. See [Localization of user interface elements](#).

Example – to set up a dialog box header with padding, 22 x 12 pixels icon, bold 16 point title, and a 28 x 28 pixel Close button that is positioned two pixels from the top and two pixels from the right of the dialog box container:

```
.s7interactivevideoviewer .s7linkdialog .s7dialogheader {
  padding: 10px;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogheader .s7dialogline {
  padding: 10px 10px 2px;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogheadericon {
  background-image: url("images/sdk/dlglink_cap.png");
  height: 12px;
  width: 22px;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogheadertext {
  font-size: 16pt;
  font-weight: bold;
  padding-left: 16px;
}
.s7interactivevideoviewer .s7linkdialog .s7closebutton {
  top: 2px;
  right: 2px;
  padding: 8px;
  width: 28px;
  height: 28px;
}
.s7interactivevideoviewer .s7linkdialog .s7closebutton[state='up'] {
  background-image: url(images/sdk/close_up.png);
}
.s7interactivevideoviewer .s7linkdialog .s7closebutton[state='over'] {
  background-image: url(images/sdk/close_over.png);
}
.s7interactivevideoviewer .s7linkdialog .s7closebutton[state='down'] {
  background-image: url(images/sdk/close_down.png);
}
.s7interactivevideoviewer .s7linkdialog .s7closebutton[state='disabled'] {
```

```
background-image:url(images/sdk/close_disabled.png);
}
```

The dialog box footer consists of a Cancel button. The footer container is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7dialogfooter
```

CSS properties of the dialog box footer

border	Border that you can use to visually separate the footer from the rest of the dialog box.
--------	--

The footer has an inner container that keeps the button. It is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7dialogbuttoncontainer
```

CSS properties of the dialog box button container

padding	Inner padding between the footer and the button.
---------	--

The Select All button is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7dialogactionbutton
```

The button is only available on desktop systems.

CSS properties of the Select All button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: The Select All button supports the *state* attribute selector, which can be used to apply different skins to different button states.

The Cancel button is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7dialogcancelbutton
```

CSS properties of the dialog box Cancel button

width	Button width.
height	Button height.
color	Button text color for each state.
background-color	Button background color for each state.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

In addition, both buttons share the same common CSS class which can contain CSS settings that are the same for other dialog box buttons:

```
.s7interactivevideoviewer .s7linkdialog .s7dialogfooter .s7button
```

CSS properties of the button

font-weight	Button font weight.
font-size	Button font size.
font-family	Button font family.
line-height	Text height inside the button. Affects vertical alignment.
box-shadow	Drop shadow.
margin-right	Right button margin.

The button tool tips can be localized. See [Localization of user interface elements](#).

Example – to set up a dialog box footer with a 64 x 34 Cancel button, having text color and background colors that are different for each button state:

```
.s7interactivevideoviewer .s7linkdialog .s7dialogfooter {
  border-top: 1px solid #909090;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogbuttoncontainer {
  padding-bottom: 6px;
  padding-top: 10px;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogfooter .s7button {
  box-shadow: 1px 1px 1px #999999;
  color: #FFFFFF;
  font-size: 9pt;
  font-weight: bold;
  line-height: 34px;
  margin-right: 10px;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogcancelbutton {
  width: 64px;
  height: 34px;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogcancelbutton[state='up'] {
  background-color: #666666;
  color: #dddddd;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogcancelbutton[state='down'] {
  background-color: #555555;
  color: #ffffff;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogcancelbutton[state='over'] {
  background-color: #555555;
  color: #ffffff;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogcancelbutton[state='disabled'] {
  background-color: #b2b2b2;
```

```
color:#dddddd;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogactionbutton {
width:82px;
height:34px;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogactionbutton[state='up'] {
background-color:#333333;
color:#dddddd;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogactionbutton[state='down'] {
background-color:#222222;
color:#cccccc;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogactionbutton[state='over'] {
background-color:#222222;
color:#cccccc;
}
.s7interactivevideoviewer .s7linkdialog .s7dialogactionbutton[state='disabled'] {
background-color:#b2b2b2;
color:#dddddd;
}
```

The main dialog area (between the header and the footer) contains dialog content. In all cases, the component manages the width of this area—it is not possible to set it in CSS. Main dialog area is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7dialogviewarea
```

CSS properties of the dialog box viewing area

height	The height of the main dialog box area. It should be specified only when the dialog box works in desktop mode. It is not applicable when the dialog box is sized to occupy the entire browser window.
background-color	The background color of the main dialog box area.
margin	Outer margin.

Example – to set up a main dialog box area to be 300 pixels height, have a 10 pixel margin, and use a white background:

```
.s7interactivevideoviewer .s7linkdialog .s7dialogviewarea {
background-color:#ffffff;
margin:10px;
height:300px;
}
```

All form content—such as labels and input fields—resides inside a container controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7dialogbody
```

CSS properties of the dialog box body

padding	Inner padding.
---------	----------------

Example – to set up form content to have 10 pixel padding:

```
.s7interactivevideoviewer .s7linkdialog .s7dialogbody {
padding: 10px;
}
```

All static labels in the dialog box form are controlled with

```
.s7interactivevideoviewer .s7linkdialog .s7dialoglabel
```

This class is not suitable for controlling the label size or position because you can apply it to texts in various places of the form user interface.

CSS properties of the dialog box label.

font-weight	Label font weight.
font-size	Label font size.
font-family	Label font family.
color	Label text color.

The dialog box labels can be localized. See [Localization of user interface elements](#).

Example – to set up all labels to be gray, bold with a nine pixel font:

```
.s7interactivevideoviewer .s7linkdialog .s7dialoglabel {
  color: #666666;
  font-size: 9pt;
  font-weight: bold;
}
```

The size of the text copy displayed on top of the link is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7dialoginputwide
```

CSS properties of the dialog box input wide field

width	Text width.
padding	Inner padding.

Example – to set text copy to be 430 pixels wide and have a 10 pixel padding in the bottom:

```
.s7interactivevideoviewer .s7linkdialog .s7dialoginputwide {
  padding-bottom: 10px;
  width: 430px;
}
```

The share link is wrapped in a container and controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7dialoginputcontainer
```

CSS properties of the dialog box input container

border	Border around the share link container.
padding	Inner padding.

Example – to set a one pixel grey border around embed code text and have nine pixels of padding:

```
.s7interactivevideoviewer .s7linkdialog .s7dialoginputcontainer {
  border: 1px solid #CCCCCC;
  padding: 9px;
}
```

The share link itself is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7linkdialog .s7dialoglink
```

CSS properties of the dialog box share link

width	Share link width.
-------	-------------------

Example – to set the share link to be 450 pixels wide:

```
.s7interactivevideoviewer .s7linkdialog .s7dialoglink {
  width: 450px;
}
```

Main viewer area

The main view area is the area occupied by the interactive swatches. It is usually set to fit the available device screen when no size is specified.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7interactivevideoviewer
```

CSS property	Description
width	The width of the viewer.
height	The height of the viewer.
background-color	Background color in hexadecimal format.

Example

To set up a viewer with a white background (#FFFFFF) and make its size 512 x 288 pixels.

```
.s7interactivevideoviewer {
  background-color: #FFFFFF;
  width: 512px;
  height: 288px;
}
```

Mutable volume

The mutable volume control initially appears as a button that lets a user mute or unmute the video player sound.

When a user rolls over the button, a slider appears that allows a user to set the volume. The mutable volume control can be sized, skinned, and positioned, relative to the control bar that contains it, by CSS.

The appearance of the mutable volume area is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7mutablevolume
```

CSS properties of the mutable volume

top	Position from the top border, including padding.
right	Position from the right border, including padding.
width	The width of the mutable volume control.
height	The height of the mutable volume control.
background-color	The color of the mutable volume control.

The mute/unmute button appearance is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7mutablevolume .s7mutebutton
```

You can control background image for each button state. The size of the button is inherited from the size of the volume control.

CSS properties of the button image

background-image	The image displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports both the `state` and `selected` attribute selectors, which can be used to apply different skins to different button states. In particular, `selected='true'` corresponds to the "muted" state and `selected='false'` corresponds to the "unmuted" state.

The vertical volume bar area is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7mutablevolume .s7verticalvolume
```

CSS properties of the vertical volume bar area

background-color	The background color of the vertical volume.
width	The width of the vertical volume.
height	The height of the vertical volume.

The track inside vertical volume control is controlled with the following CSS class selectors:

```
.s7interactivevideoviewer .s7mutablevolume .s7verticalvolume .s7track  
.s7interactivevideoviewer .s7mutablevolume .s7verticalvolume .s7filledtrack
```

CSS properties of the track inside vertical volume control

background-color	The background color of the vertical volume control.
width	Width of the vertical volume control.
height	Height of the vertical volume control.

The vertical volume knob is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7mutablevolume .s7verticalvolume .s7knob
```

CSS properties of the vertical volume control knob

background-image	Vertical volume control knob artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
width	Width of the vertical volume control knob.
height	Height of the vertical volume control knob.
left	Horizontal position of the vertical volume control knob.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Examples

To set up a mute button that is 32 x 32 pixels and positioned 6 pixels from the top, and 38 pixels from the right edge of the control bar. Display a different image for each of the four different button states when selected or not selected.

```
.s7interactivevideoviewer .s7mutablevolume {
top:6px;
right:38px;
width:32px;
height:32px;
}
.s7interactivevideoviewer .s7mutablevolume .s7mutebutton[selected='true'][state='up'] {
background-image:url(images/mute_up.png);
}
.s7interactivevideoviewer .s7mutablevolume .s7mutebutton[selected='true'][state='over'] {
background-image:url(images/mute_over.png);
}
.s7interactivevideoviewer .s7mutablevolume .s7mutebutton[selected='true'][state='down'] {
background-image:url(images/mute_down.png);
}
.s7interactivevideoviewer .s7mutablevolume .s7mutebutton[selected='true'][state='disabled'] {
background-image:url(images/mute_disabled.png);
}
.s7interactivevideoviewer .s7mutablevolume .s7mutebutton[selected='false'][state='up'] {
background-image:url(images/unmute_up.png);
}
.s7interactivevideoviewer .s7mutablevolume .s7mutebutton[selected='false'][state='over'] {
background-image:url(images/unmute_over.png);
}
.s7interactivevideoviewer .s7mutablevolume .s7mutebutton[selected='false'][state='down'] {
background-image:url(images/unmute_down.png);
}
```

```
}
.s7interactivevideoviewer .s7mutablevolume .s7mutebutton[selected='false'][state='disabled']
{
background-image:url(images/unmute_disabled.png);
}
```

The following is an example of how you can style the volume slider within the mutable volume control.

```
.s7interactivevideoviewer .s7mutablevolume .s7verticalvolume {
width:36px;
height:83px;
left:0px;
background-color:#dddddd;
}
.s7interactivevideoviewer .s7mutablevolume .s7verticalvolume .s7track {
top:11px;
left:14px;
width:10px;
height:63px;
background-color:#666666;
}
.s7interactivevideoviewer .s7mutablevolume .s7verticalvolume .s7filledtrack {
width:10px;
background-color:#ababab;
}
.s7interactivevideoviewer .s7mutablevolume .s7verticalvolume .s7knob {
width:18px;
height:10px;
left:9px;
background-image:url(images/volumeKnob.png);
}
```

Play/Pause button

The play/pause button causes the video player to play or pause the video content when a user clicks it.

You can size, skin, and position the button, relative to the control bar that contains it, by CSS.

The following CSS class selector controls the appearance of the button:

```
.s7interactivevideoviewer .s7playpausebutton
```

CSS properties of the play/pause button

top	Position from the top border, including padding.
right	Position from the right border, including padding.
left	Position from the left border, including padding.
bottom	Position from the bottom border, including padding.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.

background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
---------------------	---



Note: This button supports both the *state*, *selected*, and *replay* attribute selectors, which can be used to apply different skins to different button states. In particular, *selected='true'* corresponds to the "play" state and *selected='false'* corresponds to the "pause" state;

replay='true' is set when the video has reached the end and clicking on the button restarts playback from the beginning.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example

To set up a play/pause button that is 32 x 32 pixels; it is positioned six pixels from the top and left edge of the control bar, and displays a different image for each of the four different button states when selected or not selected.

```
.s7interactivevideoviewer .s7playpausebutton {
top:6px;
left:6px;
width:32px;
height:32px;
}
.s7interactivevideoviewer .s7playpausebutton[selected='true'][state='up'] {
background-image:url(images/playBtn_up.png);
}
.s7interactivevideoviewer .s7playpausebutton[selected='true'][state='over'] {
background-image:url(images/playBtn_over.png);
}
.s7interactivevideoviewer .s7playpausebutton[selected='true'][state='down'] {
background-image:url(images/playBtn_down.png);
}
.s7interactivevideoviewer .s7playpausebutton[selected='true'][state='disabled'] {
background-image:url(images/playBtn_disabled.png);
}
.s7interactivevideoviewer .s7playpausebutton[selected='false'][state='up'] {
background-image:url(images/pauseBtn_up.png);
}
.s7interactivevideoviewer .s7playpausebutton[selected='false'][state='over'] {
background-image:url(images/pauseBtn_over.png);
}
.s7interactivevideoviewer .s7playpausebutton[selected='false'][state='down'] {
background-image:url(images/pauseBtn_down.png);
}
.s7interactivevideoviewer .s7playpausebutton[selected='false'][state='disabled'] {
background-image:url(images/pauseBtn_disabled.png);
}
.s7interactivevideoviewer .s7playpausebutton[selected='true'][replay='true'][state='up'] {
background-image:url(images/replayBtn_up.png);
}
.s7interactivevideoviewer .s7playpausebutton[selected='true'][replay='true'][state='over'] {
background-image:url(images/replayBtn_over.png);
}
.s7interactivevideoviewer .s7playpausebutton[selected='true'][replay='true'][state='down'] {
background-image:url(images/replayBtn_down.png);
}
.s7interactivevideoviewer .s7playpausebutton[selected='true'][replay='true'][state='disabled'] {
background-image:url(images/replayBtn_disabled.png);
}
```

Social share

The social share tool appears in the top right corner by default. It consists of a button and a panel that expands when the user clicks or taps on a button and contains individual sharing tools.

The position and size of the social share tool in the viewer user interface is controlled with the following:

```
.s7interactivevideoviewer .s7socialshare
```

CSS properties of the social share tool

top	Vertical position of the social sharing tool relative to the viewer container.
left	Horizontal position of the social sharing tool relative to the viewer container.
width	The width of the social sharing tool.
height	The height of the social sharing tool.

Example

To set up a social sharing tool that is positioned four pixels from the top and five pixels from the right of viewer container and is sized 28 x 28 pixels.

```
.s7interactivevideoviewer .s7socialshare {  
  top:4px;  
  right:5px;  
  width:28px;  
  height:28px;  
}
```

The appearance of the social share tool button is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7socialshare .s7socialbutton
```

CSS properties of the social share tool button

background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the *state* attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#).

Example

To set up a social sharing tool button that displays a different image for each of the four different button states.

```
.s7interactivevideoviewer .s7socialshare .s7socialbutton[state='up'] {  
  background-image:url(images/v2/SocialShare_video_dark_up.png);  
}  
.s7interactivevideoviewer .s7socialshare .s7socialbutton[state='over'] {  
  background-image:url(images/v2/SocialShare_dark_over.png);  
}
```

```

}
.s7interactivevideoviewer .s7socialshare .s7socialbutton[state='down'] {
background-image:url(images/v2/SocialShare_dark_down.png);
}
.s7interactivevideoviewer .s7socialshare .s7socialbutton[state='disabled'] {
background-image:url(images/v2/SocialShare_dark_disabled.png);
}

```

The appearance of the panel which contains the individual social sharing icons is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7socialshare .s7socialsharepanel
```

CSS properties of the social share panel

background-color	The background color of the panel.
------------------	------------------------------------

Example

To set up a panel to have transparent color:

```

.s7interactivevideoviewer .s7socialshare .s7socialsharepanel {
background-color: transparent;
}

```

Tooltips

On desktop systems some user interface elements such as buttons have tooltips that are displayed on mouse hover.

CSS properties of the main viewer area

The appearance of tooltips is controlled with the following CSS class selector:

```
.s7tooltip
```

CSS property	Description
border-radius	Background border radius.
border-color	Background border color.
background-color	Background color.
color	Text color.
font-family	Text font name.
font-size	Text font size.



Note: In case tooltip styles are customized from within the embedding web page, all properties must contain the **! IMPORTANT** rule. This is not necessary if tooltips are customized within the viewer's CSS file.

Example

To set up tooltips that have a grey border with a three pixel corner radius, black background, and white text in Arial, 11 pixels:

```
.s7tooltip {
border-radius: 3px 3px 3px 3px;
border-color: #999999;
background-color: #000000;
color: #FFFFFF;
font-family: Arial, Helvetica, sans-serif;
font-size: 11px;
}
```

Twitter share


Twitter share tool consists of a button added to the Social share panel. When the button is clicked the user is redirected to a sharing dialog box that is provided by a social service. The position of the button is fully managed by the Social share tool.

The appearance of the Twitter share button is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7twittershare
```

CSS properties of the Twitter share tool

width	Button width.
height	Button height.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

 **Note:** This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

It is possible to remove the button from the Social share panel by setting `display:none` CSS property on its CSS class.

The button tool tip can be localized. See [Localization of user interface elements](#).

Example

To set up a Twitter share button that is 28 x 28 pixels, and displays a different image for each of the four different button states:

```
.s7interactivevideoviewer .s7twittershare {
width:28px;
height:28px;
}
.s7interactivevideoviewer .s7twittershare[state='up'] {
background-image:url(images/v2/TwitterShare_dark_up.png);
}
.s7interactivevideoviewer .s7twittershare[state='over'] {
background-image:url(images/v2/TwitterShare_dark_over.png);
}
.s7interactivevideoviewer .s7twittershare[state='down'] {
background-image:url(images/v2/TwitterShare_dark_down.png);
}
.s7interactivevideoviewer .s7twittershare[state='disabled'] {
background-image:url(images/v2/TwitterShare_dark_disabled.png);
}
```

Video player

The video player is the rectangular area where the video content is displayed within the viewer.

If the dimensions of the video that is being played does not match the dimensions of the video player, the video content is centered within the video player's rectangle display area.

The following CSS class selector controls the appearance of the video player:

```
.s7interactivevideoviewer .s7videoplayer
```

CSS properties of the video player

background-color	Background color of the main view.
------------------	------------------------------------

You can localize the error message that displayed in cases where the system is unable to play the video.

See [Localization of user interface elements](#).

Example – To set up a video viewer with the video player size set to 512 x 288 pixels.

```
.s7interactivevideoviewer .s7videoplayer{
background-color: transparent;
}
```

Closed captions are put into an internal container inside the video player. The position of that container is controlled by supported WebVTT positioning operators. The caption text itself is inside that container, and its style is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7videoplayer .s7caption
```

CSS properties of closed captioning

background-color	Closed caption text background.
color	Close caption text color.
font-weight	Closed caption font weight.
font-size	Closed caption font size.
font-family	Closed caption font.

Example

To set up closed caption text to be 14 pixels, light gray, Arial, on a semi-transparent black background:

```
.s7interactivevideoviewer .s7videoplayer .s7caption {
background-color: rgba(0,0,0,0.75);
color: #e6e6e6;
font-weight: normal;
font-size: 14px;
font-family: Arial,Helvetica,sans-serif;
}
```

The appearance of the buffering animation is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7videoplayer .s7waiticon
```

CSS properties of wait icon

CSS property	Description
width	Animation icon width.
height	Animation icon height.
margin-left	Animation icon left margin, normally minus half of the icon's width.
margin-top	Animation icon top margin, normally minus half of the icon's height.
background-image	Knob artwork.

Example – to set up a buffering animation to be 101 pixels wide, 29 pixels high:

```
.s7interactivevideoviewer .s7videoplayer .s7waiticon {
  width: 101px;
  height: 29px;
  margin-left: -50px;
  margin-top: -15px;
  background-image: url(images/sdk/busyicon.gif);
}
```

Video scrubber

The video scrubber is the horizontal slider control that lets a user dynamically seek to any time position within the currently playing video.

The scrubber 'knob' also moves as the video plays to indicate the current time position of the video during playback. The video scrubber always takes the whole width of the control bar. It is possible to skin the video scrubber. change its height and vertical position, by CSS.

The general appearance of the video scrubber is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7videoscrubber
.s7interactivevideoviewer .s7videoscrubber .s7videotime
.s7interactivevideoviewer .s7videoscrubber .s7knob
```

CSS properties of the video scrubber

top	Position from the top border, including padding.
bottom	Position from the bottom border, including padding.
height	Height of the video scrubber.

background-color	The color of the video scrubber.
------------------	----------------------------------

The following CSS class selectors track background, play, and load indicators:

```
.s7interactivevideoviewer .s7videoscrubber .s7track
.s7interactivevideoviewer .s7videoscrubber .s7trackloaded
.s7interactivevideoviewer .s7videoscrubber .s7trackplayed
```

CSS properties of the track

height	Height of the corresponding track.
background-color	The color of the corresponding track.

The following CSS class selector controls the knob:

```
.s7interactivevideoviewer .s7videoscrubber .s7knob
```

CSS properties of the knob

top	Vertical knob offset.
width	Width of knob.
height	Height of knob.
background-image	Knob artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .

The following CSS class selector controls the time played bubble:

```
.s7interactivevideoviewer .s7videoscrubber .s7videotime
```

CSS properties of the time played bubble

font-family	The font family to use for the time display text.
font-size	The font size to use for the time display text.
color	The font color to use for the time display text.
width	Bubble area width.
height	Bubble area height.
padding	Bubble area padding.

background-image	Bubble artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .
text-align	Alignment of text with the bubble area.

The video scrubber tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – To set up a video viewer with a video scrubber with custom track colors that is 10 pixels tall, and positioned 10 pixels and 35 pixels from the top and left edges of the control bar.

```
.s7interactivevideoviewer .s7videoscrubber {
top:10px;
left:35px;
height:10px;
background-color:#AAAAAA;
}
.s7interactivevideoviewer .s7videoscrubber .s7track {
height:10px;
background-color:#444444;
}
.s7interactivevideoviewer .s7videoscrubber .s7trackloaded {
height:10px;
background-color:#666666;
}
.s7interactivevideoviewer .s7videoscrubber .s7trackplayed {
height:10px;
background-color:#888888;
}
```

When video chaptering is enabled with the navigation parameter, chapter locations are displayed as markers on top of the video scrubber track.

The video chapter marker is controlled by the following CSS class selector:

```
.s7interactivevideoviewer .s7videoscrubber .s7navigation
```

CSS properties of the video chapter marker

width	Video chapter marker width.
height	Video chapter marker height.
background-image	Video chapter marker artwork.
background-position	Position inside artwork sprite, if CSS sprites are used. See CSS Sprites .



Note: This button supports the *state* attribute selector, which you can use to apply different skins to different button states. In particular, *selected='default'* corresponds to the default video chapter marker state and *selected='over'* is used when the video chapter marker is activated by a mouse over or touch gesture.

Example – To set up a video chapter marker that is 5 x 8 pixels and uses different art for the "default" and "over" state.

```
.s7interactivevideoviewer .s7videoscubber .s7navigation {
width:5px;
height:8px;
}
.s7interactivevideoviewer .s7videoscubber .s7navigation[state="default"] {
background-image: url("images/v2/VideoScrubberDiamond.png");
}
.s7interactivevideoviewer .s7videoscubber .s7navigation[state="over"] {
background-image: url("images/v2/VideoScrubberDiamond_over.png");
}
```

Video chapter bubble is positioned on top of the video chapter marker and shows the title, start time, and description for a given chapter. It is possible to control the maximum bubble width and vertical offset relative to the video scrubber track. The rest is calculated automatically by the component.

The video chapter bubble is controlled by the following CSS class selector:

```
.s7interactivevideoviewer .s7videoscubber .s7chapter
```

CSS properties of the video chapter bubble

max-width	Maximum width of the video chapter bubble.
bottom	Vertical offset from the video scrubber track.

Example – To set up a video chapter bubble that is 235 pixels wide and is eight pixels up from the bottom of the video scrubber track.

```
.s7interactivevideoviewer .s7videoscubber .s7chapter {
max-width:235px;
bottom:8px;
}
```

The video chapter bubble consists of an optional header and content. The header has the optional chapter start time and chapter title.

The header is controlled by the following CSS class selector:

```
.s7interactivevideoviewer .s7videoscubber .s7chapter .s7header
```

CSS properties of the video chapter bubble header

height	Video chapter bubble header height.
padding	Inner padding for video chapter bubble header text.
background-color	Video chapter bubble header background color.
line-height	Video chapter bubble header text line height.

Example – To set up a video chapter bubble header that is 22 pixels high, a 22 pixel line height, a 12 pixel horizontal margin, and a gray background.

```
.s7interactivevideoviewer .s7videoscubber .s7chapter .s7header {
height:22px;
padding:0 12px;
```

```
line-height:22px;
background-color: rgba(51, 51, 51, 0.8);
}
```

The start time of the video chapter is controlled by the following CSS class selector:

```
.s7interactivevideoviewer .s7videoscubber .s7chapter .s7header .s7starttime
```

CSS properties of the video chapter start time

color	Text color.
font-weight	Font weight.
font-size	Font size.
font-family	Font family.
padding-right	Padding between the start time and chapter title.

Example – To set up chapter start time using gray ten pixels Verdana font and has ten pixels padding to the right.

```
.s7interactivevideoviewer .s7videoscubber .s7chapter .s7header .s7starttime {
color: #ddddd;
font-family: Verdana,Arial,Helvetica,sans-serif;
font-size: 10px;
padding-right: 10px;
}
```

The video chapter title is controlled by the following CSS class selector:

```
.s7interactivevideoviewer .s7videoscubber .s7chapter .s7header .s7title
```

CSS properties of the video chapter title

color	Video chapter title text color.
font-weight	Video chapter title font weight.
font-size	Video chapter title font size.
font-family	Video chapter title font family.

Example – To set up a video chapter title that uses a white, bold, ten pixel Verdana font.

```
.s7interactivevideoviewer .s7videoscubber .s7chapter .s7header .s7title {
color: #ffffff;
font-family: Verdana,Arial,Helvetica,sans-serif;
font-size: 10px;
font-weight: bold;
}
```

The video chapter description is controlled by the following CSS class selector:

```
.s7interactivevideoviewer .s7videoscubber .s7chapter .s7description
```

CSS properties of the video chapter description

color	Video chapter description text color.
background-color	Video chapter description background color.
font-weight	Video chapter description font weight.
font-size	Video chapter description font size.
font-family	Video chapter description font family.
line-height	Video chapter description line height.
padding	Video chapter description inner padding.

Example – To set up video chapter description using a dark gray, 11 pixel Verdana font, with a light gray background; 5 pixel line height, 12 pixel horizontal padding, 12 pixel top padding, and 9 pixel bottom padding.

```
.s7interactivevideoviewer .s7videoscrubber .s7chapter .s7description {
color: #333333;
background-color: rgba(221, 221, 221, 0.9);
font-family: Verdana,Arial,Helvetica,sans-serif;
font-size: 11px;
line-height: 15px;
padding: 12px 12px 9px;
}
```

The wedge connector within the bottom of the chapter bubble is controlled by the following CSS class selector:

```
.s7interactivevideoviewer .s7videoscrubber .s7chapter .s7tail
```

CSS properties of the wedge connector

border-color	Wedge connector color. Defined as <color> transparent transparent so that only the top border color is defined and the remaining borders are left transparent.
border-width	Wedge connector width. Defined as <width> <width> 0 so that the same width is defined for the top and horizontal borders only and the bottom border width is 0.
margin	Defines a negative bottom margin only. It should have the same value as that of border-width.

Example – To set up a gray, six pixel wedge connector:

```
.s7interactivevideoviewer .s7videoscrubber .s7chapter .s7tail {
border-color: rgba(221, 221, 221, 0.9) transparent transparent;
border-width: 6px 6px 0;
margin: 0 0 0 -6px;
}
```

Video time

The video time is the numeric display that shows the current time and duration of the currently playing video.

The video time font family, font size, and font color are among the properties that CSS can control. It can also be positioned, relative to the control bar that contains it, by CSS.

The appearance of the video time is controlled with the following CSS class selector:

```
.s7interactivevideoviewer .s7videotime
```

CSS properties of video time

top	Position from the top border, including padding.
right	Position from the right border, including padding.
width	The width of video time control. This property is required for Internet Explorer 8 or greater to function properly.
font-family	The font family to use for the time display text.
font-size	The font size to use for the time display text.
color	The font color to use for the time display text.

Example

Set the video time to light gray (hexadecimal #BBBBBB), sized at 12 pixels, positioned 15 pixels from the top of the control bar, and 80 pixels from the top and right edges of the control bar.

```
.s7interactivevideoviewer .s7videotime {
top:15px;
right:80px;
font-size:12px;
color:#BBBBBB;
width:60px;
}
```

Support for Adobe Analytics tracking

By default, the viewer sends a single tracking HTTP request to the configured Image Server with the viewer type and version information.

Custom tracking

To integrate with third-party analytics systems it is necessary to listen to the `trackEvent` viewer callback and process the `eventInfo` argument of the callback function as necessary. The following code is an example of such handler function:

```
var interactiveVideoViewer = new s7viewers.InteractiveVideoViewer({
  "containerId":"s7viewer",
  "params":{
    "asset":"/content/dam/mac/aodmarketingna/dm-viewers-content/video/Glacier.mp4",
    "config":"/etc/dam/presets/viewer/Shoppable_Video_Dark",
```

```

"serverurl":"https://aodmarketingna.assetsadobe.com/is/image/",
"videoserverurl":"https://gateway-na.assetsadobe.com/DMGateway/public/aodmarketingna",
"contenturl":"https://aodmarketingna.assetsadobe.com/",
"interactivedata":"is/content/content/dam/mac/aodmarketingna/_VIT/dm-viewers-content/video/Glacier.mp4.svideo.vtt"
},
"handlers":{
  "trackEvent":function(objID, compClass, instName, timeStamp, eventInfo) {
    //identify event type
    var eventType = eventInfo.split(",")[0];
    switch (eventType) {
      case "LOAD":
        //custom event processing code
        break;
      case "INTERACTIVE_SWATCH":
        //custom event processing code which handles user clicks on interactive swatches
        break;
      //additional cases for other events
    }
  }
};

```

The viewer tracks the following SDK user events:

SDK user event	Sent...
LOAD	when viewer is loaded first.
SWAP	when an asset is swapped in the viewer using <code>setAsset()</code> API.
PLAY	when playback starts.
PAUSE	when playback is paused.
STOP	when playback is stopped.
MILESTONE	when playback reaches one of the following milestones: 0%, 25%, 50%, 75%, or 100%.
INTERACTIVE_SWATCH	each time the user clicks an interactive swatch.

HTTPS video delivery



Note: Secure Video Delivery only applies to AEM 6.2 with the installation of [Feature Pack-13480](#) and to AEM 6.1 with installation of [Feature Pack NPR-15011](#).

Provided that the viewer works in configuration as outlined at the beginning of this section, published video delivery can happen both in HTTPS (secure) and HTTP (insecure) modes. In a default configuration, the video delivery protocol strictly follows the delivery protocol of the embedding web page. However, it is possible to force HTTPS video delivery without regard to the protocol used by embedding the web page using the [VideoPlayer.ssl](#) configuration attribute. (Note that video preview in Author mode is always delivered securely over HTTPS.)

Depending on the method of publishing Dynamic Media video that you use in AEM, the `VideoPlayer.ssl` configuration attribute is applied differently as demonstrated in the following:

- If you publish a Dynamic Media video with a URL, you append `VideoPlayer.ssl` to the URL. For example, to force secure video delivery, you append `&VideoPlayer.ssl=on` to the end of the following viewer URL example:

~~<https://demos-pub.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/InteractiveVideoViewer.js?VideoPlayer.ssl=on>~~

See also [\(AEM 6.2\) Linking URLs to your Web Application](#) or [\(AEM 6.1\) Linking URLs to your Web Application](#)

- If you publish a Dynamic Media video with embed code, you add `VideoPlayer.ssl` to the list of other viewer configuration parameters in the embed code snippet. For example, to force HTTPS video delivery, you append `&VideoPlayer.ssl=on` as in the following example:

```
<style type="text/css">
  #s7interactivevideo_div.s7interactivevideoviewer{
    width:100%;
    height:auto;
  }
</style>
<script type="text/javascript"
src="https://demos-pub.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/InteractiveVideoViewer.js"></script>
<div id="s7interactivevideo_div"></div>
<script type="text/javascript">
  var s7interactivevideoviewer = new s7viewers.InteractiveVideoViewer({
    "containerId" : "s7interactivevideo_div",
    "params" : {
      "VideoPlayer.ssl" : "on",
      "serverurl" : "https://adobedemo62-h.assetsadobe.com/is/image",
      "contenturl" : "https://demos-pub.assetsadobe.com/",
      "config" : "/etc/dam/presets/viewer/Shoppable_Video_light",
      "config2" : "/etc/dam/presets/analytics",
      "videoserverurl": "https://gateway-na.assetsadobe.com/DMGateway/public/demoCo",
      "interactivedata":
"content/dam/_VTT/marketing/shoppable-video/adobe-axis-demo/Adobe_AXIS_V3_GRADED-HD.mp4.svideo.vtt",

      "VideoPlayer.contenturl": "https://adobedemo62-h.assetsadobe.com/is/content",
      "asset" :
"/content/dam/marketing/shoppable-video/adobe-axis-demo/Adobe_AXIS_V3_GRADED-HD.mp4" }
    })
  /* // Example of interactive video event for quick view.
  s7interactivevideoviewer.setHandlers({
    "quickViewActivate": function(inData) {
      var sku=inData.sku; //SKU for product ID
      //To pass other parameter from the hotspot, you will need to add custom parameter during
the hotspot setup as parameterName=value
      loadQuickView(sku); //Replace this call with your quickview plugin
      //Please refer to your quickviewer plugin for the quickview call
    },
    "initComplete":function() {
      //--- Attach quickview popup to viewer container so popup will work in fullscreen mode
      ---
      var popup = document.getElementById('quickview_div'); // get custom quick view container
      popup.parentNode.removeChild(popup); // remove it from current DOM
      var sdkContainerId =
s7interactivevideoviewer.getComponent("container").getInnerContainerId(); // get viewer
container component
      var inner_container = document.getElementById(sdkContainerId);
      inner_container.appendChild(popup); //Attach custom quick view container to viewer
    }
  });
  */
  s7interactivevideoviewer.init();
</script>
```

See also [\(AEM 6.2\) Embedding the Video on a Web Page](#) or [\(AEM 6.1\) Embedding the Video on a Web Page](#).

Interactive data support

The Video Video Viewer supports rendering interactive swatches based on interactive data passed to the viewer as a configuration parameter.

The currently visible swatch corresponds to the video time region with which it is associated. Clicking or tapping the interactive swatch triggers the action assigned to it at author time.

Interactive swatch can either activate a Quick View on the hosting web page by triggering a JavaScript callback or it can redirect the user to an external web page.

About Quick View

These types of interactive swatches should be authored using the action type "quickview" in AEM Assets – on-demand. When a user activates such a swatch, the viewer runs `quickViewActivate` JavaScript callback and passes the swatch data to it. It is expected that the embedding web page listens to this callback and when it triggers, the page opens its own Quick View implementation.

Redirect to an external web page

Swatches authored for the action type "quickview" in AEM Assets – on-demand redirect the user to an external URL. Depending on the settings at the time of authoring, the URL can open either in a new browser tab, in the same window, or in the named browser window.

Localization of user interface elements

Certain content that the Video Video Viewer displays is subject to localization. This includes user interface element tool tips and an error message that is displayed when the video is unable to play.

Every textual content in the viewer that can be localized is represented by the special Viewer SDK identifier called SYMBOL. Any SYMBOL has a default associated text value for an English locale ("en") supplied with the out-of-the-box viewer, and also may have user-defined values set for as many locales as needed.

When the viewer starts, it checks the current locale to see if there is a user-defined value for each supported SYMBOL for such locale. If there is, it uses the user-defined value; otherwise, it falls back to the out-of-the-box default text.

User-defined localization data can be passed to the viewer as a localization JSON object. Such object contains the list of supported locales, SYMBOL text values for each locale, and the default locale.

An example of such a localization object is the following:

```
{
  "en": {
    "VideoPlayer.ERROR": "Your Browser does not support HTML5 Video tag or the video cannot be played.",
    "PlayPauseButton.TOOLTIP_SELECTED": "Play"
  },
  "fr": {
    "VideoPlayer.ERROR": "Votre navigateur ne prend pas en charge la vidéo HTML5 tag ou la vidéo ne peuvent pas être lus.",
    "PlayPauseButton.TOOLTIP_SELECTED": "Jouer"
  },
  defaultLocale: "en"
}
```

In the example above, the localization object defines two locales ("en" and "fr") and provides localization for two user interface elements in each locale.

The web page code should pass the localization object to the viewer constructor, as a value of `localizedTexts` field of the configuration object. An alternative option is to pass the localization object by calling `setLocalizedTexts(localizationInfo)` method.

The following SYMBOLs are supported:

SYMBOL	Tool tip for...
<code>PlayPauseButton.TOOLTIP_SELECTED</code>	Selected play pause button state.
<code>PlayPauseButton.TOOLTIP_UNSELECTED</code>	Deselected play pause button state.
<code>PlayPauseButton.TOOLTIP_REPLAY</code>	Replay play pause button state.
<code>VideoScrubber.TOOLTIP</code>	Video scrubber.
<code>VideoTime.TOOLTIP</code>	Video time on control bar.
<code>MutableVolume.TOOLTIP_SELECTED</code>	Selected mutable volume.
<code>MutableVolume.TOOLTIP_UNSELECTED</code>	Deselected mutable volume.
<code>FullScreenButton.TOOLTIP_SELECTED</code>	Full screen button in normal state.
<code>FullScreenButton.TOOLTIP_UNSELECTED</code>	Full screen button in full screen state.
<code>ClosedCaptionButton.TOOLTIP_SELECTED</code>	Selected closed caption button state.
<code>ClosedCaptionButton.TOOLTIP_UNSELECTED</code>	Deselected closed caption button state.
<code>InteractiveSwatches.BANNER</code>	Caption for the banner.
<code>ScrollUpButton.TOOLTIP</code>	Scroll up button.
<code>ScrollDownButton.TOOLTIP</code>	Scroll down button.
<code>SocialShare.TOOLTIP</code>	Social share tool.
<code>LinkShare.TOOLTIP</code>	Link share button.
<code>LinkShare.HEADER</code>	Link dialog box header.
<code>LinkShare.TOOLTIP_HEADER_CLOSE</code>	Link dialog box upper-right close button.
<code>LinkShare.DESRIPTION</code>	Description of the share link.

SYMBOL	Tool tip for...
<code>LinkShare.CANCEL</code>	Caption for the Cancel button.
<code>LinkShare.TOOLTIP_CANCEL</code>	Cancel button.
<code>LinkShare.ACTION</code>	Caption for the Select All button.
<code>LinkShare.TOOLTIP_ACTION</code>	Select All button.
<code>FacebookShare.TOOLTIP</code>	Facebook share button.
<code>TwitterShare.TOOLTIP</code>	Twitter share button.
<code>CloseButton.TOOLTIP</code>	Call to action panel Close button.
<code>VideoPlayer.ERROR</code>	Error message that appears when no video playback is possible.

Full screen support

The viewer supports full-screen operation mode.

On modern desktop browsers, except Internet Explorer 10 and older, and on some touch devices, the viewer uses "native" full screen mode. This mode means that the entire device screen is occupied by the viewer content.

On iOS devices and on older Internet Explorer browsers, the viewer uses "simulated" full screen mode instead. In this mode, the viewer resizes to take the full area of the web browser window. Also, the web browser Chrome and other windows are still visible on the screen.

A user enters and leaves full screen mode by pressing the Full Screen button in the viewer user interface. When "native" full screen mode is used on desktop, it is also possible to exit it by pressing **Esc**.

Viewer SDK namespace

The viewer is built of many Viewer SDK components. In most cases, the web page does not need to interact with SDK components API directly; all common needs are covered in the viewer API itself.

However, some advanced use cases require that the web page obtain a reference to an inner SDK component using the `getComponent()` viewer API and then use all the flexibility of the APIs of SDK itself.

The namespace that is used to load and initialize SDK components by the viewer depends on the environment in which the viewer is operating. If the viewer is running in AEM (Adobe Experience Manager), the viewer loads SDK components into `s7viewers.s7sdk` namespace. Likewise, the viewer served from Scene7 Publishing System loads the SDK into `s7classic.s7sdk`.

In either case, the namespace used by the SDK inside the viewer has either `s7viewers` or `s7classic` as the prefix. And, it is different from the plain `s7sdk` namespace used in the SDK User Guide or SDK API documentation.

For that reason, it is important to use a fully qualified SDK namespace when you write custom application code that communicates with internal viewer components.

For example, if you plan to listen to `StatusEvent.NOTF_VIEW_READY` event and the viewer is served from AEM, the fully qualified event type is `s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY`, and the event listener code looks similar to the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var videoPlayer = <instance>.getComponent("videoPlayer");
    videoPlayer.addEventListener(s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e)
    {
      console.log("view ready");
    }, false);
  }
});
```

Carousel

Carousel Viewer is a viewer that displays a carousel of non-zoomable banner images with clickable hotspots or regions. The purpose of this viewer is to implement a "shoppable carousel" experience where users can select a hotspot or a region over the banner image and get redirected to a Quickview or product detail page on the customer's website. It is designed to work on desktops and mobile devices.



Note: Images that use Image Rendering or User-Generated Content (UGC) are not supported by this viewer.

Viewer type is 511.

Demo URL

https://marketing.adobe.com/resources/help/en_US/s7/viewers_ref/samples/CarouselViewerDemo.html

System Requirements

See [System requirements](#).

Using Carousel Viewer

Carousel Viewer represents a main JavaScript file and a set of helper files (a single JavaScript include with all Viewer SDK components used by this particular viewer, assets, CSS) downloaded by the viewer in runtime.

Carousel Viewer can be used both in pop-up mode using a production-ready HTML page provided with IS Viewers, or in embedded mode where it is integrated into the target web page using documented API.

Configuration and skinning are similar to that of the other viewers described in this Help. All skinning is achieved by way of custom CSS.

See [Command reference common to all viewers – Configuration attributes](#) and [Command reference common to all Viewers – URL](#)

Interacting with Carousel Viewer

Navigating through the carousel set is done using a horizontal swipe over the main view or with two arrow buttons available on the desktop device. Set indicator dots show the current position within the set.

The viewer can render hotspots or regions on top of the banner image to indicate the interactive area on the product.

Clicking or tapping on a hotspot or a region triggers an action associated with it during the author time. The action may be redirected to a different page on the web site, or it can pass product information back to the web page logic, which in turn can trigger a Quickview with related product content.

The viewer is fully keyboard accessible.

See [Keyboard accessibility and navigation](#).

Embedding Carousel Viewer

About pop-up mode

In pop-up mode, the viewer is opened in a separate web browser window or tab. It takes the entire browser window area and adjusts in case the browser is resized, or a mobile device's orientation is changed.

Pop-up mode is the most common for mobile devices. The web page loads the viewer using `window.open()` JavaScript call, properly configured A HTML element, or any other suitable method.

It is recommended that you use an out-of-the-box HTML page for pop-up operation mode. In this case, it is called `CarouselViewer.html` and is located within the `html5/` subfolder of your standard IS-Viewers deployment:

<s7viewers_root>/html5/CarouselViewer.html

You can achieve visual customization by applying custom CSS.

The following is an example of HTML code that opens the viewer in a new window:

Open popup viewer

About fixed size embedding mode and responsive design embedding mode

In the embedded mode, the viewer is added to the existing web page. This web page may already have some customer content not related to the viewer. The viewer normally occupies only a part of a web page's real estate.

The primary use cases are web pages oriented for desktops or tablet devices, and responsive designed pages that adjust layout automatically depending on the device type.

Fixed size embedding is used when the viewer does not change its size after initial load. This is the best choice for web pages that have a static layout.

Responsive design embedding assumes that the viewer may need to resize at runtime in response to the size change of its container `DIV`. The most common use case is adding a viewer to a web page that uses a flexible page layout.

In responsive design embedding mode, the viewer behaves differently depending on the way web page sizes its container `DIV`. If the web page sets only the width of the container `DIV`, leaving its height unrestricted, the viewer automatically chooses its height according to the aspect ratio of the asset used. This functionality ensures that the asset fits perfectly into the view without any padding on the sides. This use case is the most common for web pages using responsive web design layout frameworks like Bootstrap, Foundation, and so on.

Otherwise, if the web page sets both the width and the height for the viewer's container `DIV`, the viewer fills just that area. It also follows the size that the web page layout provides. A good example is embedding the viewer into a modal overlay, where the overlay is sized according to web browser window size.

Fixed size embedding

You add the viewer to a web page by doing the following:

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Setting the viewer size.
4. Creating and initializing the viewer.

1. Adding the viewer JavaScript file to your web page.

Creating a viewer requires that you add a script tag in the HTML head. Before you can use the viewer API, be sure that you include `CarouselViewer.js`. The `CarouselViewer.js` file is located under the `html5/js/` subfolder of your standard IS-Viewers deployment:

```
<s7viewers_root>/etc/dam/viewers/s7viewers/html5/js/CarouselViewer.js
```

You can use a relative path if the viewer is deployed on one of the Adobe Scene7 servers and it is served from the same domain. Otherwise, you specify a full path to one of Adobe Scene7 servers that have the IS-Viewers installed.

The relative path looks like the following:

```
<script language="javascript" type="text/javascript"
src="/etc/dam/viewers/s7viewers/html5/js/CarouselViewer.js"></script>
```



Note: You should only reference the main viewer JavaScript include file on your page. You should not reference any additional JavaScript files in the web page code which might be downloaded by the viewer's logic in runtime. In particular, do not directly reference `HTML5 SDK Utils.js` library loaded by the viewer from `/s7viewers` context path (so-called consolidated SDK include). The reason is that the location of `Utils.js` or similar runtime viewer libraries is fully managed by the viewer's logic and the location changes between viewer releases. Adobe does not keep older versions of secondary viewer includes on the server.

As a result, putting a direct reference to any secondary JavaScript include used by the viewer on the page breaks the viewer functionality in the future when a new product version is deployed.

2. Defining the container DIV.

Add an empty DIV element to the page where you want the viewer to appear. The DIV element must have its ID defined because this ID is passed later to the viewer API. The DIV has its size specified through CSS.

The placeholder DIV is a positioned element, meaning that the `position` CSS property is set to `relative` or `absolute`.

The following is an example of a defined placeholder DIV element:

```
<div id="s7viewer" style="position:relative"></div>
```

3. Setting the viewer size

You can set the static size for the viewer by either declaring it for `.s7carouselviewer` top-level CSS class in absolute units, or by using `stagesize` modifier.

You can put sizing in CSS directly on the HTML page, or in a custom viewer CSS file, which is then later assigned to a viewer preset record in AEM Assets – on-demand, or passed explicitly using the `style` command.

See [Customizing Carousel Viewer](#) for more information about styling the viewer with CSS.

The following is an example of defining a static viewer size in the HTML page:

```
#s7viewer.s7carouselviewer {
width: 1174px;
```

```
height: 500px;
}
```

You can pass explicitly the `stagesize` modifier with the viewer initialization code with `params` collection or as an API call as described in the Command Reference section, like this:

```
carouselViewer.setParam("stagesize", "1174,500");
```

A CSS-based approach is recommended and is used in this example.

4. Creating and initializing the viewer.

When you have completed the steps above, you create an instance of `s7viewers.CarouselViewer` class, pass all configuration information to its constructor, and call `init()` method on a viewer instance. Configuration information is passed to the constructor as a JSON object. At minimum, this object should have `containerId` field which holds the name of viewer container ID and nested `params` JSON object with configuration parameters supported by the viewer. In this case, the `params` object must have at least the Image Serving URL passed as `serverUrl` property, and the initial asset as `asset` parameter. The JSON-based initialization API lets you create and start the viewer with a single line of code.

It is important to have the viewer container added to the DOM so that the viewer code can find the container element by its ID. Some browsers delay building DOM until the end of the web page. For maximum compatibility, call the `init()` method just before the closing `BODY` tag, or on the `body onload()` event.

At the same time, the container element should not necessarily be part of the web page layout just yet. For example, it may be hidden using `display:none` style assigned to it. In this case, the viewer delays its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load automatically resumes.

The following is an example of creating a viewer instance, passing minimum necessary configuration options to the constructor and calling the `init()` method. The example assumes `carouselViewer` is the viewer instance; `s7viewer` is the name of placeholder DIV; `https://adobedemo62-h.assetsadobe.com/is/image` is the Image Serving URL, and `/content/dam/dm-public-facing-live-demo-page/04_shoppable_carousel/05_shoppable_banner` is the asset:

```
<script type="text/javascript">
var carouselViewer = new s7viewers.CarouselViewer ({
  "containerId": "s7viewer",
  "params": {

    "asset": "/content/dam/dm-public-facing-live-demo-page/04_shoppable_carousel/05_shoppable_banner",

    "serverurl": "https://adobedemo62-h.assetsadobe.com/is/image"
  }
}).init();
</script>
```

The following code is a complete example of a trivial web page that embeds the Carousel Viewer with a fixed size:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="https://demos-pub.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/CarouselViewer.js"></script>
<style type="text/css">
#s7viewer.s7carouselviewer {
  width: 1174px;
  height: 500px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative"></div>
```

```

<script type="text/javascript">
var carouselViewer = new s7viewers.CarouselViewer({
  "containerId": "s7viewer",
  "params": {

    "asset": "/content/dam/dm-public-facing-live-demo-page/04_shoppable_carousel/05_shoppable_banner",

    "serverurl": "https://adobedemo62-h.assetsadobe.com/is/image"
  }
}).init();
</script>
</body>
</html>

```

Responsive design embedding with unrestricted height

With responsive design embedding, the web page normally has some kind of flexible layout in place that dictates the runtime size of the viewer's container DIV. For the following example, assume that the web page allows the viewer's container DIV to take 40% of the web browser window size. And, its height is left unrestricted. The web page HTML code would look like the following:

```

<!DOCTYPE html>
<html>
<head>
<style type="text/css">
.holder {
  width: 80%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>

```

Adding the viewer to such a page is similar to the steps for fixed size embedding. The only difference is that you do not need to explicitly define the viewer size.

1. Adding the viewer JavaScript file to your web page.
2. Defining the container DIV.
3. Creating and initializing the viewer.

All the steps above are the same as with the fixed size embedding. Add the container DIV to the existing "holder" DIV. The following code is a complete example. Notice how the viewer size changes when the browser is resized, and how the viewer aspect ratio matches the asset.

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="https://demos-pub.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/CarouselViewer.js"></script>
<style type="text/css">
.holder {
  width: 80%;
}
</style>
</head>
<body>
<div class="holder">
<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var carouselViewer = new s7viewers.CarouselViewer({

```



```

    "containerId": "s7viewer",
    "params": {
      "asset": "/content/dam/dm-public-facing-live-demo-page/04_shoppable_carousel/05_shoppable_banner",
      "serverurl": "https://adobedemo62-h.assetsadobe.com/is/image"
    }
  }).init();
</script>
</body>
</html>

```

The following examples page illustrates more real-life uses of responsive design embedding with unrestricted height:

https://marketing.adobe.com/resources/help/en_US/s7/viewers_ref/samples/CarouselViewer-responsive-unrestricted-height.html

Flexible size Embedding with Width and Height Defined

In case of flexible-size embedding with width and height defined, the web page styling is different. It provides both sizes to the "holder" DIV and center it in the browser window. Also, the web page sets the size of the HTML and BODY element to 100 percent.

```

<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
</style>
</head>
<body>
<div class="holder"></div>
</body>
</html>

```

The rest of the embedding steps are identical to the steps used for responsive embedding with unrestricted height. The resulting example is the following:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="https://demos-pub.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/CarouselViewer.js"></script>
<style type="text/css">
html, body {
  width: 100%;
  height: 100%;
}
.holder {
  position: absolute;
  left: 20%;
  top: 20%;
  width: 60%;
  height: 60%;
}
</style>
</head>
<body>
<div class="holder">

```

```

<div id="s7viewer" style="position:relative"></div>
</div>
<script type="text/javascript">
var carouselViewer = new s7viewers.CarouselViewer({
  "containerId":"s7viewer",
  "params":{

"asset":"/content/dam/dm-public-facing-live-demo-page/04_shoppable_carousel/05_shoppable_banner",

  "serverurl":"https://adobedemo62-h.assetsadobe.com/is/image"
  }
}).init();
</script>
</body>
</html>

```

Embedding Using Setter-based API

Instead of using JSON-based initialization, it is possible to use setter-based API and no-args constructor. Using this API constructor does not take any parameters and configuration parameters are specified using `setContainerId()`, `setParam()`, and `setAsset()` API methods with separate JavaScript calls.

The following example illustrates using fixed size embedding with the setter-based API:

```

<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="https://demos-pub.assetsadobe.com/etc/dam/viewers/s7viewers/html5/js/CarouselViewer.js"></script>
<style type="text/css">
#s7viewer.s7carouselviewer {
  width: 1174px;
  height: 500px;
}
</style>
</head>
<body>
<div id="s7viewer" style="position:relative"></div>
<script type="text/javascript">
var carouselViewer = new s7viewers.CarouselViewer();
carouselViewer.setContainerId("s7viewer");
carouselViewer.setParam("serverurl", "https://adobedemo62-h.assetsadobe.com/is/image");
carouselViewer.setAsset("/content/dam/dm-public-facing-live-demo-page/04_shoppable_carousel/05_shoppable_banner");
carouselViewer.init();
</script>
</body>
</html>

```

Command reference – Configuration attributes

Configuration attributes documentation for Carousel Viewer.

Any configuration command can be set in URL or using `setParam()`, or `setParams()`, or both, API methods. Any config attribute can be also specified in the server-side configuration record.

Some configuration commands may be prefixed with the class name or instance name of corresponding Viewer SDK component. An instance name of the component is dynamic and depends on the ID of the viewer container DOM element passed to `setContainerId()` API method. Documentation includes an optional prefix for such commands. For example, `zoomstep` command is documented as follows:

```
[ZoomView.|<containerId>_carouselView].fmt
```

which means that you can use this command as:

- `fmt` (short syntax)
- `CarouselView.fmt` (qualified with component class name)
- `cont_carouselView.fmt` (qualified with component ID, assuming `cont` is the ID of the container element)

See also [Command reference common to all viewers – Configuration attributes](#)

CarouselView.autoplay

Configuration attribute for Carousel Viewer.

`[CarouselView.|<containerId>_carouselView.]autoplay=[0|1][,duration][,direction]`

<code>[0 1][,duration][,direction]</code>	<p>Specifies on/off, duration to display each banner in the carousel and direction of auto-loop.</p> <p>Set to 0 for auto-loop off.</p> <p>Set 1 to auto-loop on with transition duration in seconds controlled by <code>duration</code>.</p> <p>The direction of auto-loop is controlled with <code>direction</code>. The <code>direction</code> has the range between 1 right-to-left and 0 left-to-right.</p>
---	--

Properties

Optional.

Default

1,9

Example

`autoplay=1,2,1`

CarouselView.frametransition

`[CarouselView.|<containerId>_carouselView.]frametransition=none|fade|slide[,duration[,spacing]`

<code>none fade slide</code>	<p>Specifies the type of the effect applied on frame change. <code>none</code> stands for no transition; frame change happens instantly.</p> <p><code>fade</code> means cross-fade transition between old and new frames.</p> <p><code>slide</code> activates transition where the old frame slides out of the view and the new frame slides in.</p>
<code>duration</code>	<p>Specifies the duration (in second) of <code>fade</code> or <code>slide</code> transition effect.</p>
<code>spacing</code>	<p>The spacing between adjacent frames in <code>slide</code> transition, has the range between 0 and 1 and is relative to component's width.</p>

Properties

Optional.

Default

None.

Example

```
frametransition=fade,0.3
```

CarouselView.iscommand

```
[CarouselView. | <containerId>_carouselView. ]iscommand=isCommand
```

<i>iscommand</i>	The Image Serving command string that is applied to the banner image. If specified in the URL all occurrences of & and = must be HTTP-encoded as %26 and %3D, respectively.
------------------	---

Properties

Optional.

Default

None.

Example

When specified in the viewer URL:

```
iscommand=op_sharpen%3d1%26op_colorize%3d0xff0000
```

When specified in the config data:

```
iscommand=op_sharpen=1&op_colorize=0xff0000
```

CarouselView.maxloadradius

```
[CarouselView. | <containerId>_carouselView. ]maxloadradius=-1 | 0 | preloadnbr
```

<i>-1 0 preloadnbr</i>	<p>Specifies the component preload behavior.</p> <p>When set to -1 the component will preload all carousel frames when in an idle state.</p> <p>When set to 0 the component loads only the frame that is currently visible, previous, and next frame.</p> <p><i>preloadnbr</i> defines how many invisible frames around the currently displayed frame are preloaded when in an idle state.</p>
----------------------------	--

Properties

Optional.

Default

1

Example

maxloadradius=0

CarouselView.enableHD

[CarouselView. | <containerId>_carouselView.]enableHD=always|never|limit[,number]

always never limit	<p>Enable, limit or disable optimization for devices where devicePixelRatio is greater than 1, that is devices with high-density display like iPhone4 and similar devices.</p> <p>If active then the component limits the size of the IS image request as if the device only had a pixel ratio of 1 and that way reducing the bandwidth.</p> <p>See example below.</p>
number	<p>If using the limit setting, the component enables high pixel density only up to the specified limit.</p> <p>See example below.</p>

Properties

Optional.

Default

limit,1500

Example

enableHD=always

CarouselView.fmt

[CarouselView. | <containerId>_carouselView.]fmt=jpg|jpeg|png|png-alpha|gif|gif-alpha

jpg jpeg png png-alpha gif gif-alpha	<p>Specifies the image format that the component uses for loading images from Image Server.</p> <p>If the specified format ends with -alpha, the component renders images as transparent content.</p> <p>For all other image formats the component treats images as opaque. The component has a white background by default. Therefore, to make it transparent, set the background-color CSS property to transparent.</p>
--------------------------------------	---

Properties

Optional.

Default

jpeg

Example

fmt=png-alpha

SetIndicator.autohide

[SetIndicator.<containerId>_setIndicator.]autohide=0|1[,limit]

0 1[,limit]	<p>Configures auto-hide behavior depending on number of pages and run-time component size.</p> <p>0 turns the auto-hide off.</p> <p>1 enables the auto-hide. The component hides its dots if at least one of the following conditions turn true:</p> <ul style="list-style-type: none">• the row with dots becomes wider than the run-time component width, or• number of pages set for this component exceeds the limit configured by the <i>limit</i> parameter. <p>Setting <i>limit</i> to -1 disables the second auto-hide condition.</p>
-------------	--

Properties

Optional.

Default

1,10

Example

autohide=0

SetIndicator.mode

[SetIndicator.<containerId>_setIndicator.]mode=numeric|dotted

numeric dotted	<p>Configures the rendering style of the set indicator.</p> <p>When set to <i>dotted</i> the component renders identical indicators for all pages.</p> <p>When set to <i>numeric</i> it puts a 1-based page number inside each indicator element.</p> <p>The <i>numeric</i> operation mode is not supported on devices that are capable of touch input. Instead, the component uses <i>dotted</i> on such devices.</p>
----------------	--

Properties

Optional.

Default

dotted

Example

mode=numeric

ControlBar.transition

Configuration attribute for Carousel Viewer.

[ControlBar.<containerId>_controlBar.]transition=none|fade[,delaytohide[,duration]]

none fade	Specifies the effect type that is used to show or hide the control bar and its content. Set to none for instant show/hide. Set to fade to provide a gradual fade in/out effect.
delaytohide	Specifies the time in seconds between the last mouse/touch event registered by the control bar and the time control bar hides. If set to -1 the component never triggers its auto-hide effect and, therefore, always stays visible on the screen.
duration	Sets the duration of the fade in/out animation in seconds.

Properties

Optional. This command is ignored on touch devices where the control bar auto-hide is disabled.

Default

fade, 2, 0.3

Example

transition=none

Command reference – URL

Command reference documentation for Carousel Viewer.

preloadimage**

URL command for Carousel Viewer.

preloadImage=0 | 1

0 1	Enable (1) or disable (0) the preload image feature. See Preload image .
-------	---

Properties

Optional.

Default

1

Example

```
preloadImage=0
```

JavaScript API reference for Carousel Viewer

The main class of the Carousel Viewer is `CarouselViewer`. It is declared in the `s7viewers` namespace. This JavaScript API covers constructor, methods, and callbacks of this particular class.

In all the following examples, `<instance>` stands for the actual name of the JavaScript viewer object that is instantiated from the `s7viewers.CarouselViewer` class.

CarouselViewer

JavaScript API reference for Carousel Viewer.

```
CarouselViewer([config])
```

Constructor, creates a new HTML 5 Carousel Viewer instance.

Parameters

<i>config</i>	<p><code>{object}</code> optional JSON configuration object, allows all the viewer settings to pass to the constructor to avoid calling individual setter methods. Contains the following properties:</p> <ul style="list-style-type: none">• <code>containerId</code> – <code>{String}</code> ID of the DOM container (normally a <code>DIV</code>) that the viewer is inserted into. By the time this method is called, it is not necessary to have the container element created. However, the container must exist when <code>init()</code> is run. <p>Required.</p> <ul style="list-style-type: none">• <code>params</code> – <code>{Object}</code> JSON object with viewer configuration parameters where the property name is either viewer-specific configuration option or SDK modifier, and the value of that property is a corresponding settings value. <p>Required.</p> <ul style="list-style-type: none">• <code>handlers</code> – <code>{Object}</code> JSON object with viewer event callbacks, where the property name is the name of supported viewer event and the property value is a JavaScript function reference to appropriate callback. <p>Optional.</p> <p>See Event callbacks for more information about viewer events.</p> <ul style="list-style-type: none">• <code>localizedTexts</code> – <code>{Object}</code> <p>JSON object with localization data. See Localization of user interface elements and the example for more information about the object's content.</p>
---------------	--

Optional

Returns

None.

Example

```
var carouselViewer = new s7viewers.CarouselViewer({
  "containerId": "s7viewer",
  "params": {

    "asset": "/content/dam/dm-public-facing-live-demo-page/04_shoppable_carousel/05_shoppable_banner",

    "serverurl": "https://adobedemo62-h.assetsadobe.com/is/image"
  },
  "handlers": {
    "initComplete": function() {
      console.log("init complete");
    }
  },
  "localizedTexts": {
    "en": {
      "PanLeftButton.TOOLTIP": "Left"
    },
    "fr": {
      "PanLeftButton.TOOLTIP": "Gauchiste"
    }
  },
  defaultLocale: "en"
});
```

dispose

JavaScript API reference for Carousel Viewer.

`dispose()`

Disposes this viewer instance by releasing all resources used by the viewer logic and deleting all inner objects and components created by the viewer in runtime.

The web page code should also delete the viewer instance variable as well to completely remove the viewer from the web browser memory.

If the web page code has registered event listeners directly on Viewer SDK components used by the viewer—or stored external references to such components—such listeners must be explicitly unregistered by the web page code, and such external component references must be deleted prior to calling `dispose()`.

Do not access the Viewer API any more after `dispose()` is called.

Parameters

None.

Returns

None.

Example

```
<instance>.dispose()
```

getComponent**

JavaScript API reference for Carousel Viewer.

```
getComponent(componentId)
```

Returns a reference to the Viewer SDK component that is used by the viewer. The web page can use this method to extend or customize the behavior of the out-of-box viewer. Call this method only after the `initComplete` viewer callback has run, otherwise the component may not be created yet by the viewer logic.

Parameters

componentID - {String} an ID of the Viewer SDK component used by the viewer. This viewer supports the following component IDs:

Component ID	Viewer SDK component class name
parameterManager	s7sdk.ParameterManager
container	s7sdk.common.Container
mediaSet	s7sdk.set.MediaSet
carouselView	s7sdk.set.CarouselView
imageMapEffect	s7sdk.image.mageMapEffect
controlBar	s7sdk.common.ControlBar
setIndicator	s7sdk.set.SetIndicator
nextButton	s7sdk.common.PanRightButton
prevButton	s7sdk.common.PanLeftButton

When working with SDK APIs it is important to use the correct fully qualified SDK namespace as described in [Viewer SDK namespace](#).

Refer to the Viewer SDK API documentation for more information about a particular component.

Returns

{Object} a reference to Viewer SDK component. The method returns `null` if the `componentId` is not a supported viewer component or if the component was not yet created by the viewer logic.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    var carouselView = <instance>.getComponent( "carouselView" );
  }
})
```

init

JavaScript API reference for Carousel Viewer.

`init()`

Starts the initialization of the Carousel Viewer. By this time the container DOM element must be created so that the viewer code can find it by its ID.

If the container element is not a part of the web page layout just yet (for example, it may be hidden using `display:none` style assigned to it), the viewer suspends its initialization process until the moment when the web page brings the container element back to the layout. When this happens, the viewer load automatically resumes.

Call this method only once during viewer life cycle; subsequent calls are ignored.

Parameters

None.

Returns

{Object} A reference to the viewer instance.

Example

```
<instance>.init()
```

setAsset

JavaScript API reference for Carousel Viewer.

`setAsset(asset)`

Sets the new asset. You can call this parameter at any time, either before or after `init()`. If it is called after `init()`, the viewer swaps the asset at runtime.

See also [init](#).

<i>asset</i>	<code>{String}</code> new asset ID. Images which use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.
--------------	--

Returns

None.

Example

Single image reference:

```
<instance>.setAsset("/content/dam/dm-public-facing-live-demo-page/04_shoppable_carousel/05_shoppable_banner")
```

setContainerId

JavaScript API reference for Carousel Viewer.

```
setContainerId(containerId)
```

Sets the ID of the DOM container (normally a `DIV`) into which the viewer is inserted. It is not necessary to have the container element created by the time this method is called. However, the container must exist when `init()` is run. It must be called before `init()`.

This method is optional if the viewer configuration information is passed with the `config` JSON object to the constructor.

Parameter

<i>containerId</i>	{string} ID of container.
--------------------	---------------------------

Returns

None.

Example

```
<instance>.setContainerId("s7viewer");
```

setHandlers

JavaScript API reference for Carousel Viewer

```
setHandlers(handlers)
```

Specifies zero or more callback handlers. A call to this method fully overwrites event handlers that were previously assigned for that viewer instance. Must be called before `init()`.

Parameter

<i>handlers</i>	<p>{Object} JSON object with viewer event callbacks. The property name is the name of the supported viewer event. The property value is a JavaScript function reference to an appropriate callback.</p> <p>See Event callbacks for more information about viewer events.</p>
-----------------	--

Returns

None.

Example

```
<instance>.setHandlers({
  "initComplete":function() {
    console.log("init complete");
  }
});
```

```
}  
})
```

setLocalizedTexts

JavaScript API reference for Carousel Viewer.

```
setLocalizedTexts( localizationInfo )
```

Sets localization SYMBOL values for one or more locales. This parameter must be called before `init()`.

<i>localizationInfo</i>	<p>{Object} JSON object with localization data.</p> <p>See Localization of user interface elements for more information.</p> <p>See also the <i>Viewer SDK User Guide</i> and the example for more information about the object's content.</p>
-------------------------	--

See also [init](#).

Returns

None.

Example

```
<instance>.setLocalizedTexts({ "en": { "PanLeftButton.TOOLTIP": "Left" }, "fr": { "PanLeftButton.TOOLTIP": "Gauchiste" }, defaultLocale: "en" })
```

setParam

JavaScript API reference for Carousel Viewer.

```
setParam(name, value)
```

Sets the viewer parameter to a specified value. The parameter is either a viewer-specific configuration option or a software development kit modifier. This parameter is called before `init()`.

This method is optional if the viewer configuration information was passed with `config` JSON object to constructor.

See also.

Parameters

<i>name</i>	{string} name of parameter.
<i>value</i>	{string} value of parameter. The value cannot be percent-encoded.

Returns

None.

Example

```
<instance>.setParam("style", "etc/dam/presets/css/html5_carouselviewer_dotted_light.css")
```

setParams

JavaScript API reference for Carousel Viewer.

```
setParams(params)
```

Sets one or more parameters to a given value. The method argument syntax is identical to a URL query string. That is, it represents name=value pairs separated with &. Just as in a query string, the names and values are percent-encoded using UTF8. Before you call `init()`, this parameter must be called.

This method is optional if the viewer configuration information was passed with `config` JSON object to constructor.

See also [init](#).

Parameter

<i>params</i>	{string} name=value parameter pairs separated with &.
---------------	---

Returns

None.

Example

```
<instance>.setParams("CarouselView.fmt=png&CarouselView.iscommand=op_sharpen%3d1")
```

Event callbacks

The viewer supports JavaScript event callbacks that the web page uses to track the viewer initialization process or runtime behavior.

Callback handlers are assigned by passing event names and corresponding handler functions with the `handlers` property to `config` JSON object in the viewer's constructor. Alternatively, it is possible to use `setHandlers()` API method.

Supported viewer events include the following:

Viewer event	Description
<code>initComplete</code>	Triggers when viewer initialization is complete and all internal components are created, so that it is possible to use <code>getComponent()</code> API. The callback handler does not take any arguments.
<code>trackEvent</code>	Triggers each time an event occurs inside the viewer which may be handled by an event tracking system, such as Adobe Analytics. The callback handler accepts the following arguments: <ul style="list-style-type: none">• <code>objID</code> {String} – not currently used.• <code>compClass</code> {String} – not currently used.• <code>instName</code> {String} – an instance name of the Viewer SDK component that triggered the event.• <code>timeStamp</code> {Number} – event time stamp.• <code>eventInfo</code> {String} – event payload.

Viewer event	Description
quickViewActivate	<p>Triggers when the user activates a hotspot with Quick View data associated with it. The callback handler takes the following argument:</p> <ul style="list-style-type: none"> • <code>data {Object}</code> – a JSON object containing data from the hotspot definition. The field <code>sku</code> is mandatory while other fields are optional and depend on the source hotspot definition.

See also [CarouselViewer](#) and [setHandlers](#).

Customizing Carousel Viewer

All visual customization and most behavior customization for the Carousel Viewer is done by creating a custom CSS.

The suggested workflow is to take the default CSS file for the appropriate viewer, copy it to a different location, customize it, and specify the location of the customized file in the `style=` command.

Default CSS files can be found at the following location:

```
<s7viewers_root>/etc/dam/viewers/s7viewers/html5/CarouselViewer_dotted_light.css
```

The viewer is supplied with four out-of-the-box CSS files, for numeric and dotted set indicator, each in a "light" and "dark" color scheme. The "Dotted light" version is used by default, but it is easy to switch to a different version by using different standard CSS and setting the `SetIndicator.mode` parameter. Other standard CSS are found at the following location:

```
<s7_viewers_root>/html5/CarouselViewer_dotted_dark.css
```

```
<s7_viewers_root>/html5/CarouselViewer_numeric_dark.css
```

```
<s7_viewers_root>/html5/CarouselViewer_numeric_light.css
```

Custom CSS file must contain the same class declarations as the default one. If a class declaration is omitted, the viewer does not function properly because it does not provide built-in default styles for user interface elements.

An alternative way to provide custom CSS rules is to use embedded styles directly on the web page or in one of linked external CSS rules.

When creating custom CSS keep in mind that the viewer assigns `.s7carouselviewer` class to its container DOM element. If you are using an external CSS file passed with the `style=` command, use `.s7carouselviewer` class as parent class in descendant selector for your CSS rules. If you are adding embedded styles on the web page, additionally qualify this selector with an ID of the container DOM element as follows:

```
#<containerId>.s7carouselviewer
```

Building responsive designed CSS

It is possible to target different devices and embedding sizes in CSS to make your content display differently, depending on a user's device or a particular web page layout. This includes, but is not limited to, different layouts, user interface element sizes, and artwork resolution.

The viewer supports two mechanisms of creating responsive designed CSS: CSS markers and standard CSS media queries. You can use these independently or together.

CSS markers

To assist in creating responsive designed CSS, the viewer supports CSS markers. These are special CSS classes that are dynamically assigned to the top-level viewer container element. They are based on the runtime viewer size and the input type used on the current device.

The first group of CSS markers contains `.s7size_large`, `.s7size_medium`, and `.s7size_small` classes. They are applied based on the runtime area of the viewer container. For example, if the viewer area is equal or bigger than the size of a common desktop monitor, use `.s7size_large`. If the area is close to a common tablet device, assign `.s7size_medium`. For areas similar to mobile phone screens, use `.s7size_small`. The primary purpose of these CSS markers is to create different user interface layouts for different screens and viewer sizes.

The second group of CSS markers contains `.s7mouseinput` and `.s7touchinput`. The CSS marker `.s7touchinput` is set if the current device is capable of touch input. Otherwise, `.s7mouseinput` is used. These markers are mostly intended to create user interface input elements with different screen sizes for different input types, because normally touch input requires larger elements.

The following sample CSS sets the zoom in button size to 28 x 28 pixels on systems with mouse input and to 56 x 56 pixels on touch input devices. If the viewer is sized even smaller, it is set to 20 x 20 pixels.

```
.s7carouselviewer.s7mouseinput .s7imagemapeffect .s7icon {
    width:28px;
    height:28px;
}
.s7carouselviewer.s7touchinput .s7imagemapeffect .s7icon {
    width:56px;
    height:56px;
}
.s7carouselviewer.s7size_small .s7imagemapeffect .s7icon {
    width:20px;
    height:20px;
}
```

To target devices with different pixel density you need to use CSS media queries. The following media query block would contain CSS specific to high-density screens:

```
@media screen and (-webkit-min-device-pixel-ratio: 1.5)
{
}
```

Using CSS markers is the most flexible way of building responsive designed CSS because it lets you target not only the device screen size but the actual viewer size, which is useful for responsive design layouts.

You can use the default viewer CSS file as an example of a CSS markers approach.

CSS media queries

You can also accomplish device sensing by using pure CSS media queries. Everything enclosed within a given media query block is applied only when it is run on a corresponding device.

When applied to Mobile Viewers use four CSS media queries, defined in your CSS, in the order listed below:

1. Contains only rules specific for all touch devices.

```
@media only screen and (max-device-width:13.5in) and (max-device-
height:13.5in) and (max-device-width:799px),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in)
and (max-device-height:799px)
{
}
```


2. Contains only rules specific for tablets with high-resolution screens.

```
@media only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-width:799px) and (-webkit-min-device-pixel-ratio:1.5),
only screen and (max-device-width:13.5in) and (max-device-height:13.5in) and
(max-device-height:799px) and (-webkit-min-device-pixel-ratio:1.5)
{
}
```

3. Contains only rules specific for all mobile phones.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in)
{
}
```

4. Contains only rules specific for mobile phones with high-resolution screens.

```
@media only screen and (max-device-width:9in) and (max-device-height:9in) and
(-webkit-min-device-pixel-ratio: 1.5),
only screen and (device-width:720px) and (device-height:1280px) and
(-webkit-device-pixel-ratio: 2),
only screen and (device-width:1280px) and (device-height:720px) and
(-webkit-device-pixel-ratio: 2)
{
}
```

Using a media queries approach, you should organize CSS with device sensing as follows:

- First, the desktop-specific section defines all properties that are either desktop-specific or common to all screens.
- And second, the four media queries go in the order defined above and provide CSS rules that are specific for the corresponding device type.

There is no need to duplicate the entire viewer CSS in each media query. Only properties that are specific to given devices are redefined inside a media query.

CSS sprites

Many viewer user interface elements are styled using bitmap artwork and have more than one distinct visual state. A good example is a button that normally has at least three different states: up, over, and down. Each state requires its own bitmap artwork assigned.

With a classic approach to styling, the CSS would have a separate reference to the individual image file on the server for each state of the user interface element. The following is a sample CSS for styling a zoom-in button:

```
.s7carouselviewer .s7imagemapeffect .s7icon {
background-image: url(images/v2/imagemap/ImageMapEffect_icon1_light_up_touch.png);
}
.s7carouselviewer .s7imagemapeffect .s7icon:hover {
background-image: url(images/v2/imagemap/ImageMapEffect_icon1_light_over_touch.png);
}
```

The drawback to this approach is that the end user experiences flickering or delayed user interface response when the element is interacted with for the first time. This action occurs because the image artwork for the new element state is not yet downloaded. Also, this approach may have a slight negative impact on performance because of an increase in the number of HTTP calls to the server.

CSS sprites is a different approach where image artwork for all element states is combined into a single PNG file called a "sprite". Such "sprite" has all visual states for the given element positioned one after another. When styling a user interface element with sprites the same sprite image is referenced for all different states in the CSS. Also, the `background-position` property is used

for each state to specify which part of the "sprite" image is used. You can structure a "sprite" image in any suitable way. Viewers normally have it vertically stacked.

The following is a "sprite"-based example of styling the same hot spot icon:

```
.s7carouselviewer .s7imagemapeffect .s7icon {
background-image: url(images/v2/imagemap/ImageMapEffect_icon1_light_up_touch.png);
background-position: -0px -56px; width: 56px; height: 56px;
}
.s7carouselviewer .s7imagemapeffect .s7icon: hover {
background-position: -0px -0px; width: 56px; height: 56px;
}
```

General styling notes and advice

- When customizing the viewer user interface with CSS the use of the `!important` rule is not supported to style viewer elements. In particular, `!important` rule should not be used to override any default or runtime styling provided by the viewer or Viewer SDK. The reason for this is that it may affect the behavior of proper components. Instead, you should use CSS selectors with the proper specificity to set CSS properties that are documented in this Viewers Reference guide.
- All paths to external assets within CSS are resolved against the CSS location, not the viewer HTML page location. Be aware of this rule when you copy the default CSS to a different location. Either copy the default assets as well or update all the paths within the custom CSS.
- The preferred format for bitmap artwork is PNG.
- Bitmap artwork is assigned to user interface elements using the `background-image` property.
- The `width` and `height` properties of a user interface element define its logical size. The size of the bitmap passed to `background-image` does not affect its logical size.
- To use the high pixel density of high-resolution screens like Retina, specify bitmap artwork twice as large as the logical user interface element size. Then, apply the `-webkit-background-size: contain` property to scale the background down to the logical user interface element size.
- To remove a button from the user interface, add `display: none` to its CSS class.
- You can use various formats for color value that CSS supports. If you need transparency, use the format `rgba(R, G, B, A)`. Otherwise, you can use the format `#RRGGBB`.

Common user interface elements

The following is user interface elements reference documentation that applies to the Video Image Viewer:

Carousel view

Main view consists of the banner image.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7carouselviewer .s7carouselview
```

CSS property	Description
<code>background-color</code>	Background color in hexadecimal format of the main view.

Example – to make the main view transparent.

```
.s7carouselviewer .s7carouselview {
  background-color: transparent;
}
```

Focus highlight

Input focus highlight displayed around focused viewer user interface element is controlled with the CSS class selector.

CSS properties

The appearance is controlled with the following CSS class selector:

```
.s7carouselviewer *:focus
```

CSS property	Description
outline	Focus highlight style.

Example – to disable the default browser focus highlight for all viewer user interface elements, add the following CSS selector to viewer's style sheet:

```
.s7carouselviewer *:focus {
  outline: none;
}
```

Hotspots and Image maps

The viewer displays hotspot icons over the main view in places where hotspots were originally authored in Dynamic Media of AEM Assets – on-demand.

CSS properties of the main viewer area

The appearance of the hotspot icon is controlled with the following CSS class selector:

```
.s7interactiveimage .s7imagemapeffect .s7icon
```

CSS property	Description
background-image	Hotspot icon artwork.
background-position	Position inside the artwork sprite, if CSS sprites are use. See CSS sprites .
width	Hotspot icon width.
height	Hotspot icon height.

Example – set up a 56 x 56 pixels hotspot icon that displays a different image for each of the two different icon states:

```
.s7interactiveimage .s7imagemapeffect .s7icon {
  background-image: url(images/v2/imagemap/ImageMapEffect_icon1_light_up_touch.png);
  width: 56px;
  height: 56px;
}
```

```
.s7interactiveimage .s7imagemapeffect .s7icon:hover {
  background-image: url(images/v2/imagemap/ImageMapEffect_icon1_light_over_touch.png);
}
```

CSS properties of the image map region

The appearance of the image map region is controlled with the following CSS class selector:

```
.s7carouselviewer .s7imagemapeffect .s7region
```

CSS property	Description
background	Image map region fill color. Specify this color in #RRGGBB, RGB(R,G,B), or RGBA(R,G,B,A) formats.
background-color	Image map region fill color. Specify this color in #RRGGBB, RGB(R,G,B), or RGBA(R,G,B,A) formats.
border	Image map region border style. Should be specified as " <i>width solid color</i> ", where <i>width</i> is expressed in pixels, and <i>color</i> is set as #RRGGBB, RGB(R,G,B), or RGBA(R,G,B,A).

Example – set up a transparent image map region with a one pixel black border:

```
.s7carouselviewer .s7imagemapeffect .s7region {
  border: 1px solid #000000;
  background: RGBA(0,0,0,0);
}
```

Main viewer area

The main view area is the area occupied by the carousel banner image. It is usually set to fit the available device screen when no size is specified.

CSS properties of the main viewer area

The appearance of the viewing area is controlled with the following CSS class selector:

```
.s7carouselviewer
```

CSS property	Description
width	The width of the viewer.
height	The height of the viewer.
background-color	Background color in hexadecimal format.

Example – to set up a viewer with a white background (#FFFFFF) and make its size 1174 x 500 pixels.

```
.s7carouselviewer {
  background-color: #FFFFFF;
  width: 1174px;
}
```

```
height: 500px;
}
```

Next slide

Clicking or tapping on this button moves a user to the next slide in the carousel set. This button is not displayed on touch devices. You can size, skin, and position this button using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7ecatalogsearchviewer .s7panrightbutton
```

CSS property	Description
top	Position from the top of the viewer border.
right	Position from the right of the viewer border.
left	Position from the left of the viewer.
bottom	Position from the bottom of the viewer border.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS sprites .
cursor	Cursor type.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a previous slide button that is 60 x 60 pixels, positioned 10 pixels from the right viewer border and vertically centered, and displays a different image for each of the four different button states.

```
.s7carouselviewer .s7panrightbutton{
  bottom: 50%;
  right: 10px;
  background-size:60px;
  cursor: pointer;
}
.s7carouselviewer.s7mouseinput .s7panrightbutton {
  width:60px;
  height:60px;
  margin-bottom: -20px;
```

```

}
.s7carouselviewer.s7mouseinput .s7panrightbutton[state] {
  background-image:
url(../../viewers/s7viewers/html5/images/v2/CarouselDotsRightButton_dark_sprite.png);
}

.s7carouselviewer.s7mouseinput .s7panrightbutton[state='up'] { background-position: -0px -60px;
}
.s7carouselviewer.s7mouseinput .s7panrightbutton[state='over'] { background-position: -0px
-0px; }
.s7carouselviewer.s7mouseinput .s7panrightbutton[state='down'] { background-position: -0px
-0px; }
.s7carouselviewer.s7mouseinput .s7panrightbutton[state='disabled'] { background-position: -0px
-60px; }

```

Previous slide

Clicking or tapping on this button returns a user to the previous slide in the carousel set. This button is not displayed on touch devices. You can size, skin, and position this button using CSS.

CSS properties of the main viewer area

The appearance of the button is controlled with the following CSS class selector:

```
.s7carouselviewer .s7panleftbutton
```

CSS property	Description
top	Position from the top of the viewer border.
right	Position from the right of the viewer border.
left	Position from the left of the viewer border.
bottom	Position from the bottom of the viewer border.
width	Width of the button.
height	Height of the button.
background-image	The image that is displayed for a given button state.
background-position	Position inside artwork sprite, if CSS sprites are used. See also CSS sprites .
cursor	Cursor type.



Note: This button supports the `state` attribute selector, which can be used to apply different skins to different button states.

The button tool tip can be localized. See [Localization of user interface elements](#) for more information.

Example – to set up a previous slide button that is 60 x 60 pixels, positioned 10 pixels from the left viewer border and vertically centered, and displays a different image for each of the four different button states.

```
.s7carouselviewer .s7panleftbutton {
  bottom: 50%;
  left: 10px;
  background-size: 60px;
  cursor: pointer;
}
.s7carouselviewer.s7mouseinput .s7panleftbutton {
  width: 60px;
  height: 60px;
  margin-bottom: -20px;
}
.s7carouselviewer.s7mouseinput .s7panleftbutton[state] {
  background-image:
url(../../viewers/s7viewers/html5/images/v2/CarouselDotsLeftButton_dark_sprite.png);
}

.s7carouselviewer.s7mouseinput .s7panleftbutton[state='up'] { background-position: -0px -60px;
}
.s7carouselviewer.s7mouseinput .s7panleftbutton[state='over'] { background-position: -0px -0px;
}
.s7carouselviewer.s7mouseinput .s7panleftbutton[state='down'] { background-position: -0px -0px;
}
.s7carouselviewer.s7mouseinput .s7panleftbutton[state='disabled'] { background-position: -0px
-60px; }
```

Set indicator

Set indicator is a series of dots rendered at the bottom of the viewer. It shows the current position within the set.

CSS properties of the set indicator

The appearance of the set indicator container is controlled with the following CSS class selector:

```
.s7carouselviewer .s7setindicator
```

CSS property	Description
background-color	The background color in hexadecimal format of the set indicator.



Note: Set indicator supports the *mode* attribute selector, which you can use to apply different styles for dotted and numeric operation modes. In particular, *mode="numeric"* corresponds to the numeric operation mode; *mode="dotted"* corresponds to the default dot state.

Example – to set up set indicator with a white background:

```
.s7carouselviewer .s7setindicator {
  background-color: #FFFFFF;
}
```

The appearance of an individual set indicator dot is controlled with the CSS class selector. It applies to items both in dotted and numeric operation modes.

```
.s7carouselviewer .s7setindicator .s7dot
```

CSS property	Description
width	Width of the set indicator dot.
height	Height of the set indicator dot.
margin-left	Left margin in pixels.
margin-top	Top margin in pixels.
margin-right	Right margin in pixels.
margin-bottom	Bottom margin in pixels.
border-radius	Border radius in pixels.
background-color	Background color in hexadecimal format.
font-family	Name of font.
font-size	Size of font.
color	Color of font.
vertical-align	Vertical alignment of the banner index.
line-height	Text height for the banner index.



Note: Set indicator items support the *state* attribute selector, which can be used to apply different skins to different thumbnail states. In particular, *state="selected"* corresponds to the current element in the set; *state="unselected"* corresponds to the default item state.

Example – to set up set indicator in dotted mode for desktop systems to be positioned 20 pixels from the bottom of the viewer. Unselected dots are black with 50% transparency, 15 x 15 pixels with 7 pixels rounded corners. Selected dots are black with 90% transparency, 18 x 18 pixels with 9 pixels rounded corners. Spacing between dots is 5 pixels.

```
.s7carouselviewer.s7mouseinput .s7setindicator {
  bottom: 20px;
}
.s7carouselviewer.s7mouseinput .s7setindicator[mode='dotted'] .s7dot{
  width:15px;
  height:15px;
  margin-left:2.5px;
  margin-right:2.5px;
  margin-top:2.5px;
  margin-bottom:2.5px;
}
.s7carouselviewer.s7mouseinput .s7setindicator[mode='dotted'] .s7dot[state='selected'] {
  width:18px;
```



```

height:18px;
background-color: #000000;
opacity: 0.9;
border-radius:9px;
}
.s7carouselviewer.s7mouseinput .s7setindicator[mode='dotted'] .s7dot[state='unselected'] {
width:15px;
height:15px;
background-color: #000000;
opacity: 0.5;
border-radius:7px;
}

```

Tooltips

On desktop systems some user interface elements like buttons have tool tips that display on mouse hover.

CSS properties of the main viewer area

The appearance of tool tips is controlled with the following CSS class selector:

```
.s7tooltip
```

CSS property	Description
border-radius	Background border radius.
border-color	Background border color.
background-color	Background color.
color	Text color.
font-family	Text font name.
font-size	Text font size.



Note: In case tool tip styles are customized from within the embedding web page, all properties have to contain !IMPORTANT rule. This is not necessary if tool tips are customized in the viewer's CSS file.

Example – to set up tool tips that have a gray border with a 3 pixel corner radius, black background, and white text in Arial, 11 pixels size:

```

.s7tooltip {
border-radius: 3px 3px 3px 3px;
border-color: #999999;
background-color: #000000;
color: #FFFFFF;
font-family: Arial, Helvetica, sans-serif;
font-size: 11px;
}

```

Support for Adobe Analytics tracking

Custom tracking

By default, the viewer sends a single tracking HTTP request to configured Image Server with the viewer type and version information.

To integrate with third-party analytics systems, it is necessary to listen to the `trackEvent` viewer callback and process the `eventInfo` argument of the callback function as necessary. The following code is an example of such handler function:

```
var carouselViewer = new s7viewers.CarouselViewer({
  "containerId": "s7viewer",
  "params": {

    "asset": "/content/dam/dm-public-facing-live-demo-page/04_shoppable_carousel/05_shoppable_banner",

    "serverurl": "https://adobedemo62-h.assetsadobe.com/is/image"
  },
  "handlers": {
    "trackEvent": function(objID, compClass, instName, timeStamp, eventInfo) {
      //identify event type
      var eventType = eventInfo.split(",")[0];
      switch (eventType) {
        case "LOAD":
          //custom event processing code
          break;
          //additional cases for other events
      }
    }
  }
});
```

The viewer tracks the following SDK user events:

SDK user event	Sent when...
LOAD	the viewer is loaded first.
BANNER	the carousel banner image is changed.
HREF	the user activates the hotspot.

Localization of user interface elements

Certain content that the Carousel Viewer displays is subject to localization. This includes slide navigation buttons.

Every textual content in the viewer that can be localized is represented by the special Viewer SDK identifier called SYMBOL. Any SYMBOL has a default associated text value for an English locale ("en") supplied with the out-of-the-box viewer, and also may have user-defined values set for as many locales as needed.

When the viewer starts, it checks the current locale to see if there is a user-defined value for each supported SYMBOL for such locale. If there is, it uses the user-defined value; otherwise, it falls back to the out-of-the-box default text.

User-defined localization data can be passed to the viewer as a localization JSON object. Such object contains the list of supported locales, SYMBOL text values for each locale, and the default locale.

An example of such a localization object is the following:

```
{
  "en": {
    "PanLeftButton.TOOLTIP": "Left",
    "PanRightButton.TOOLTIP": "Right"
  },
  "fr": {
    "PanLeftButton.TOOLTIP": "Gauchiste",
    "PanRightButton .TOOLTIP": "Droit"
  },
  defaultLocale: "en"
}
```

In the example above, the localization object defines two locales ("en" and "fr") and provides localization for two user interface elements in each locale.

The web page code should pass the localization object to the viewer constructor, as a value of `localizedTexts` field of the configuration object. An alternative option is to pass the localization object by calling `setLocalizedTexts(localizationInfo)` method.

The following SYMBOLs are supported:

SYMBOL	Tool tip for...
PanLeftButton.TOOLTIP	Previous slide button.
PanRightButton.TOOLTIP	Next slide button.

Hotspot and Image maps support

The viewer supports the rendering of hotspot icons and image map regions on top of the main view. The appearance of hotspot icons and regions is controlled through CSS as described in the customize Hotspots and Image maps section.

See [Hotspots and Image maps](#).

Hotspots and regions can either activate a Quick View feature on the hosting web page by triggering a JavaScript callback or redirect a user to an external web page.

Quick View hotspots

These types of hotspots or image maps should be authored using the "Quick View" action type in Dynamic Media, of AEM. When a user activates such a hotspot or image map, the viewer runs the `quickViewActivate` JavaScript callback and passes the hotspot or image map data to it. It is expected that the embedding web page listens for this callback. When it triggers the page, it opens its own Quick View implementation.

Redirect to external web page

Hotspots or image maps authored for the action type "Quick View" in Dynamic Media of AEM redirects the user to an external URL. Depending on settings made during authoring, the URL opens in a new browser tab, in the same window, or in the named browser window.

Preload image

Preload image is a static asset preview image which loads right after calling `init()` method and shows while Viewer SDK libraries, asset and preset information is downloaded. The purpose of the preload image is to visually improve viewer load time and present content to the user quickly.

Preload image works well for the most common viewer embedding method, which is responsive embedding with unrestricted height. See the heading [Responsive design embedding with unrestricted height](#).

The feature, however, has certain limitations when other embedding methods or specific configuration options are used. Preload image may fail to render properly in following cases:

- When the viewer is fixed in size and the size is defined either using `stageSize` configuration attribute inside the viewer preset record or in the external viewer CSS file for the top-level viewer container element.
- When the flexible size embedding with width and height defined method of viewer embedding is used. See the heading [Flexible size embedding with width and height defined](#).

You may need to disable preload image feature using the `preloadImage` configuration attribute if you are using the viewer in one of the operation modes listed above.

Also, preload image is not used—even if enabled in the configuration—if the viewer is embedded into the DOM element is hidden using `display:none` CSS setting or detached from the DOM tree.

Viewer SDK namespace

The viewer is built of many Viewer SDK components. In most cases, the web page does not need to interact with SDK components API directly; all common needs are covered in the viewer API itself.

However, some advanced use cases require that the web page obtain a reference to an inner SDK component using the `getComponent()` viewer API and then use all the flexibility of the APIs of SDK itself.

The namespace that is used to load and initialize SDK components by the viewer depends on the environment in which the viewer is operating. If the viewer is running in AEM (Adobe Experience Manager), the viewer loads SDK components into `s7viewers.s7sdk` namespace. And the viewer served from Scene7 Publishing System loads the SDK into `s7classic.s7sdk`.

In either case, the namespace used by the SDK inside the viewer has either `s7viewers` or `s7classic` as the prefix. And, it is different from the plain `s7sdk` namespace used in the SDK User Guide or SDK API documentation.

For that reason it is important to use a fully qualified SDK namespace when you write custom application code that communicates with internal viewer components.

For example, if you plan to listen to `StatusEvent.NOTF_VIEW_READY` event and the viewer is served from AEM, the fully qualified event type is `s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY`, and the event listener code looks similar to the following:

```
<instance>.setHandlers({
  "initComplete":function() {
    var carouselView = <instance>.getComponent("carouselView");
    carouselView.addEventListener(s7viewers.s7sdk.event.StatusEvent.NOTF_VIEW_READY, function(e)
    {
      console.log("view ready");
    }, false);
  }
});
```

Command reference common to all viewers – Configuration attributes

Configuration attributes common to all viewers.

stageSize



Note: This attribute applies to the Zoom Viewer, Video Viewer, Spin Viewer, Interactive Image Viewer, Interactive Video Viewer, and Carousel Viewer.

`stageSize=width,height`

<code>width,height</code>	The overall width and height of the viewer. The value of 0,0 means that the viewer is sized according to the CSS rules and web page layout.
---------------------------	---

Properties

Optional.

Default

None.

Example

`stageSize=500,500`

style

You can apply the following command both from the URL query string and config. The command applied in the URL query string always takes precedence over the same command present in config.

`style=cssPath`

<code>cssPath</code>	Relative or absolute CSS location. Specifies the location of the custom CSS file. If the <code>cssPath</code> is relative, it is resolved against the viewer HTML page location and the value of <code>contentUrl=</code> parameter.
----------------------	---

All asset references within the CSS file are resolved against the CSS file location, not the location of the calling HTML page.

Properties

Optional.

Default

None.

Examples

```
style=/skins/customStyle.css
```

```
style=etc/dam/presets/css/html5_interactiveimage.css
```

```
style=etc/dam/presets/css/html5_interactivevideo.css
```

```
style=etc/dam/presets/css/html5_carouselviewer_dotted_light.css
```

title

```
title=windowTitle
```



Note: This command does not apply to Flyout Viewer, Basic Zoom Viewer, Spin Viewer, or Interactive Image Viewer.

<code>windowTitleimage</code>	Specifies the title of the browser window that contains the viewer.
-------------------------------	---

Properties

Optional.

Default

Adobe Scene7 Video or Adobe Scene7 eCatalog

Properties

```
title=Video
```

```
title=eCatalog
```

Command reference common to all Viewers – URL

Command reference documentation that is common to all Viewers.

The commands listed below are applied either through URL or using API `setParam()` or `setParams()` methods.

asset

Parameter common to all viewers.

`asset=assetId`

<code>imageId</code>	The asset ID of the single video or Adaptive Video Set.
----------------------	---

This property is required, unless `video` parameter is used. See [External Video Support](#).


or



`asset=image`

<code>image</code>	Specifies a single image or a carousel set. Apply double HTTP encoding to any unsafe character that exists in the image name or the carousel set name.
--------------------	--

or




`asset= image | imageList | imageListWithModifiers | multiDimensionalSpinSet [%3Fmodifiers]`



<code>image</code>	<p>Specifies a single image. Apply double HTTP encoding to any unsafe character that exists in the image name.</p> <p>Or, specifies a reference to an image set. The viewer retrieves image sets from the server, using <code>req=set IS</code> request.</p>
<code>imageList</code>	<p>Specifies an explicit image set, consisting of a sorted sequence of items, or frames, separated by commas.</p> <p> Note: This feature is supported in Adobe Scene7 Publishing System; it is not supported in Adobe Experience Manager Assets.</p>
<code>imageListWithModifiers</code>	<p>Specifies an explicit image set where each frame has its own Image Serving modifiers. In this case, the list of frames are wrapped within parentheses. Make sure you apply double HTTP encoding to any comma that is present in the frame-specific Image Serving modifier.</p>


	 Note: This feature is supported in Adobe Scene7 Publishing System; it is not supported in Adobe Experience Manager Assets.
<code>multiDimensionalSpinSet</code>	<p>Specifies an explicit multi-dimensional spin set using the following syntax:</p> <pre>((horizontalSpinSet)[,(horizontalSpinSet)])</pre> <p>where <code>horizontalSpinSet</code> is a comma-separated list of frames for a given horizontal axis. All <code>horizontalSpinSet</code> should have the same number of frames.</p>  Note: This feature is supported in Adobe Scene7 Publishing System; it is not supported in Adobe Experience Manager Assets.
<code>modifiers</code>	Image Serving commands; & and = separators must be HTTP-encoded as %26 and %3D, respectively.

or

```
asset=(mediaSet | (video;swatchId | image;swatchId | setId;swatchId);ID;(video;swatchId | image;swatchId | setId;swatchId);ID;] [%3Fmodifiers]
```

<code>mediaSet</code>	Specifies a reference to a media set. The viewer retrieves media sets from the server, using req=set IS request.
<code>video</code>	<p>Single video or Adaptive Video Set.</p>  Note: This feature is supported in Adobe Scene7 Publishing System; it is not supported in Adobe Experience Manager Assets.
<code>image</code>	<p>Single image.</p>  Note: This feature is supported in Adobe Scene7 Publishing System; it is not supported in Adobe Experience Manager Assets.
<code>setId</code>	<p>Swatch set.</p>  Note: This feature is supported in Adobe Scene7 Publishing System; it is not supported in Adobe Experience Manager Assets.
<code>swatchId</code>	Swatch image.

	 Note: This feature is supported in Adobe Scene7 Publishing System; it is not supported in Adobe Experience Manager Assets.
<i>ID</i>	<p>Media Set item type identifier can be one of the following:</p> <ul style="list-style-type: none"> • <code>advanced_image</code> For single image. • <code>advanced_swatchset</code> For nested swatch set. • <code>spin</code> For spin set. • <code>video</code> For single video. • <code>video_set</code> For Adaptive Video Sets. <p> Note: This feature is supported in Adobe Scene7 Publishing System; it is not supported in Adobe Experience Manager Assets.</p>
<i>modifiers</i>	Image Serving commands; & and = separators must be HTTP-encoded as %26 and %3D, respectively.

 **Note:** Images that use IR (Image Rendering) or UGC (User-Generated Content) are not supported by this viewer.

Properties

Required.

Default

None.

Examples

Single asset reference:

```
asset=Scene7SharedAssets/Backpack_B
```

or

```
asset=/content/dam/mac/aodmarketingna/shoppable-banner/shoppable-banner.jpg
```

or

```
asset=/content/dam/mac/aodmarketingna/dm-viewers-content/video/Glacier.mp4
```

or

```
asset=/content/dam/dm-public-facing-live-demo-page/04_shoppable_carousel/05_shoppable_banner
```

Single reference to an image set that is defined in a catalog:

```
asset=Viewers/Pluralist
```

Explicit image set:

```
asset=Scene7SharedAssets/Backpack_B,Scene7SharedAssets/Backpack_C
```

Explicit image set with frame-specific Image Serving modifiers:

```
asset=(Scene7SharedAssets/Backpack_B%3Fop_colorize%3D255%252C0%252C0,Scene7SharedAssets/Backpack_B%3Fop_colorize%3D0x00ff00)
```

Single reference to a spin set that is defined in a catalog:

```
asset=Scene7SharedAssets/SpinSet-Sample
```

Explicit spin set:

```
asset=Scene7SharedAssets/Spin1,Scene7SharedAssets/Spin2,Scene7SharedAssets/Spin3,Scene7SharedAssets/Spin4,Scene7SharedAssets/Spin5,Scene7SharedAssets/Spin6,Scene7SharedAssets/Spin7,Scene7SharedAssets/Spin8
```

Explicit multi-dimensional spin set:

```
asset=Scene7SharedAssets/Spin1,Scene7SharedAssets/Spin2,Scene7SharedAssets/Spin3,Scene7SharedAssets/Spin4,Scene7SharedAssets/Spin5,Scene7SharedAssets/Spin6,Scene7SharedAssets/Spin7,Scene7SharedAssets/Spin8,Scene7SharedAssets/Spin9,Scene7SharedAssets/Spin10,Scene7SharedAssets/Spin11,Scene7SharedAssets/Spin12,Scene7SharedAssets/Spin13,Scene7SharedAssets/Spin14,Scene7SharedAssets/Spin15,Scene7SharedAssets/Spin16,Scene7SharedAssets/Spin17,Scene7SharedAssets/Spin18,Scene7SharedAssets/Spin19,Scene7SharedAssets/Spin20,Scene7SharedAssets/Spin21,Scene7SharedAssets/Spin22,Scene7SharedAssets/Spin23,Scene7SharedAssets/Spin24,Scene7SharedAssets/Spin25,Scene7SharedAssets/Spin26,Scene7SharedAssets/Spin27,Scene7SharedAssets/Spin28,Scene7SharedAssets/Spin29,Scene7SharedAssets/Spin30,Scene7SharedAssets/Spin31,Scene7SharedAssets/Spin32,Scene7SharedAssets/Spin33,Scene7SharedAssets/Spin34,Scene7SharedAssets/Spin35,Scene7SharedAssets/Spin36,Scene7SharedAssets/Spin37,Scene7SharedAssets/Spin38,Scene7SharedAssets/Spin39,Scene7SharedAssets/Spin40,Scene7SharedAssets/Spin41,Scene7SharedAssets/Spin42,Scene7SharedAssets/Spin43,Scene7SharedAssets/Spin44,Scene7SharedAssets/Spin45,Scene7SharedAssets/Spin46,Scene7SharedAssets/Spin47,Scene7SharedAssets/Spin48,Scene7SharedAssets/Spin49,Scene7SharedAssets/Spin50,Scene7SharedAssets/Spin51,Scene7SharedAssets/Spin52,Scene7SharedAssets/Spin53,Scene7SharedAssets/Spin54,Scene7SharedAssets/Spin55,Scene7SharedAssets/Spin56,Scene7SharedAssets/Spin57,Scene7SharedAssets/Spin58,Scene7SharedAssets/Spin59,Scene7SharedAssets/Spin60,Scene7SharedAssets/Spin61,Scene7SharedAssets/Spin62,Scene7SharedAssets/Spin63,Scene7SharedAssets/Spin64,Scene7SharedAssets/Spin65,Scene7SharedAssets/Spin66,Scene7SharedAssets/Spin67,Scene7SharedAssets/Spin68,Scene7SharedAssets/Spin69,Scene7SharedAssets/Spin70,Scene7SharedAssets/Spin71,Scene7SharedAssets/Spin72,Scene7SharedAssets/Spin73,Scene7SharedAssets/Spin74,Scene7SharedAssets/Spin75,Scene7SharedAssets/Spin76,Scene7SharedAssets/Spin77,Scene7SharedAssets/Spin78,Scene7SharedAssets/Spin79,Scene7SharedAssets/Spin80,Scene7SharedAssets/Spin81,Scene7SharedAssets/Spin82,Scene7SharedAssets/Spin83,Scene7SharedAssets/Spin84,Scene7SharedAssets/Spin85,Scene7SharedAssets/Spin86,Scene7SharedAssets/Spin87,Scene7SharedAssets/Spin88,Scene7SharedAssets/Spin89,Scene7SharedAssets/Spin90,Scene7SharedAssets/Spin91,Scene7SharedAssets/Spin92,Scene7SharedAssets/Spin93,Scene7SharedAssets/Spin94,Scene7SharedAssets/Spin95,Scene7SharedAssets/Spin96,Scene7SharedAssets/Spin97,Scene7SharedAssets/Spin98,Scene7SharedAssets/Spin99,Scene7SharedAssets/Spin100
```

Single mixed media set reference:

```
asset=Scene7SharedAssets/Mixed_Media_Set-Sample
```

Explicit mixed media set:

```
asset=Scene7SharedAssets/MixedMediaSet1,Scene7SharedAssets/MixedMediaSet2,Scene7SharedAssets/MixedMediaSet3,Scene7SharedAssets/MixedMediaSet4,Scene7SharedAssets/MixedMediaSet5,Scene7SharedAssets/MixedMediaSet6,Scene7SharedAssets/MixedMediaSet7,Scene7SharedAssets/MixedMediaSet8,Scene7SharedAssets/MixedMediaSet9,Scene7SharedAssets/MixedMediaSet10,Scene7SharedAssets/MixedMediaSet11,Scene7SharedAssets/MixedMediaSet12,Scene7SharedAssets/MixedMediaSet13,Scene7SharedAssets/MixedMediaSet14,Scene7SharedAssets/MixedMediaSet15,Scene7SharedAssets/MixedMediaSet16,Scene7SharedAssets/MixedMediaSet17,Scene7SharedAssets/MixedMediaSet18,Scene7SharedAssets/MixedMediaSet19,Scene7SharedAssets/MixedMediaSet20,Scene7SharedAssets/MixedMediaSet21,Scene7SharedAssets/MixedMediaSet22,Scene7SharedAssets/MixedMediaSet23,Scene7SharedAssets/MixedMediaSet24,Scene7SharedAssets/MixedMediaSet25,Scene7SharedAssets/MixedMediaSet26,Scene7SharedAssets/MixedMediaSet27,Scene7SharedAssets/MixedMediaSet28,Scene7SharedAssets/MixedMediaSet29,Scene7SharedAssets/MixedMediaSet30,Scene7SharedAssets/MixedMediaSet31,Scene7SharedAssets/MixedMediaSet32,Scene7SharedAssets/MixedMediaSet33,Scene7SharedAssets/MixedMediaSet34,Scene7SharedAssets/MixedMediaSet35,Scene7SharedAssets/MixedMediaSet36,Scene7SharedAssets/MixedMediaSet37,Scene7SharedAssets/MixedMediaSet38,Scene7SharedAssets/MixedMediaSet39,Scene7SharedAssets/MixedMediaSet40,Scene7SharedAssets/MixedMediaSet41,Scene7SharedAssets/MixedMediaSet42,Scene7SharedAssets/MixedMediaSet43,Scene7SharedAssets/MixedMediaSet44,Scene7SharedAssets/MixedMediaSet45,Scene7SharedAssets/MixedMediaSet46,Scene7SharedAssets/MixedMediaSet47,Scene7SharedAssets/MixedMediaSet48,Scene7SharedAssets/MixedMediaSet49,Scene7SharedAssets/MixedMediaSet50,Scene7SharedAssets/MixedMediaSet51,Scene7SharedAssets/MixedMediaSet52,Scene7SharedAssets/MixedMediaSet53,Scene7SharedAssets/MixedMediaSet54,Scene7SharedAssets/MixedMediaSet55,Scene7SharedAssets/MixedMediaSet56,Scene7SharedAssets/MixedMediaSet57,Scene7SharedAssets/MixedMediaSet58,Scene7SharedAssets/MixedMediaSet59,Scene7SharedAssets/MixedMediaSet60,Scene7SharedAssets/MixedMediaSet61,Scene7SharedAssets/MixedMediaSet62,Scene7SharedAssets/MixedMediaSet63,Scene7SharedAssets/MixedMediaSet64,Scene7SharedAssets/MixedMediaSet65,Scene7SharedAssets/MixedMediaSet66,Scene7SharedAssets/MixedMediaSet67,Scene7SharedAssets/MixedMediaSet68,Scene7SharedAssets/MixedMediaSet69,Scene7SharedAssets/MixedMediaSet70,Scene7SharedAssets/MixedMediaSet71,Scene7SharedAssets/MixedMediaSet72,Scene7SharedAssets/MixedMediaSet73,Scene7SharedAssets/MixedMediaSet74,Scene7SharedAssets/MixedMediaSet75,Scene7SharedAssets/MixedMediaSet76,Scene7SharedAssets/MixedMediaSet77,Scene7SharedAssets/MixedMediaSet78,Scene7SharedAssets/MixedMediaSet79,Scene7SharedAssets/MixedMediaSet80,Scene7SharedAssets/MixedMediaSet81,Scene7SharedAssets/MixedMediaSet82,Scene7SharedAssets/MixedMediaSet83,Scene7SharedAssets/MixedMediaSet84,Scene7SharedAssets/MixedMediaSet85,Scene7SharedAssets/MixedMediaSet86,Scene7SharedAssets/MixedMediaSet87,Scene7SharedAssets/MixedMediaSet88,Scene7SharedAssets/MixedMediaSet89,Scene7SharedAssets/MixedMediaSet90,Scene7SharedAssets/MixedMediaSet91,Scene7SharedAssets/MixedMediaSet92,Scene7SharedAssets/MixedMediaSet93,Scene7SharedAssets/MixedMediaSet94,Scene7SharedAssets/MixedMediaSet95,Scene7SharedAssets/MixedMediaSet96,Scene7SharedAssets/MixedMediaSet97,Scene7SharedAssets/MixedMediaSet98,Scene7SharedAssets/MixedMediaSet99,Scene7SharedAssets/MixedMediaSet100
```

Sharpening modifier added to all images in the set:


```
asset=Scene7SharedAssets/Backpack_B,Scene7SharedAssets/Backpack_C%3Fop_sharpen=%3D1
```

```
asset=Scene7SharedAssets/Mixed_Media_Set-Sample%3Fop_sharpen%3D1
```

```
asset=Viewers/Pluralist%3Fop_sharpen%3D1
```

caption

Parameter common to all viewers.

 **Note:** This command does not apply to Video Image Viewer.

```
caption=file[,0|1]
```

file	Specifies a URL or path to the WebVTT caption content. Image Serving serves up the WebVTT file.
0 1	Specifies the default caption state. Enabled is 1.

This viewer supports closed captioning by way of hosted WebVTT files. Captions specified with this parameter apply to the video that comes first in media sets; subsequent videos play without captions. Overlapping cues and regions are not supported. Supported cue positioning operators:

Key	Name	Values	Description
A	test align	left right middle start end	Controls the alignment of text. Default is middle.
T	text position	0%–100%	Percentage of inset into the VideoPlayer component for the beginning of the caption text. Default is 0%.
S	line size	0%–100%	Percentage of video width used for captions. Default is 100%.
L	line position	0%–100% integer	Determines the line position on the page. If it is expressed as an integer with no percent sign, then it is the number of lines from the top where the text is displayed. If it is expressed as a percentage–the percent sign is the last character– then the caption text is displayed that percent down the display area. Default is 100%.

Be aware that if there are any other WebVTT features present within the WebVTT file, they are not supported; however, they will not disrupt captioning.

<i>file</i>	Specifies an URL or path to WebVTT caption content. The WebVTT file is served by Image Serving.
0 1	Specifies the default caption state. Enabled is 1.

Properties

Optional.

Default

None.

Example

```
caption=Scene7SharedAssets/adobe_qbc_final_cc,1
```

config

Parameter common to all viewers.

`config=configId`

<code>configId</code>	<p>Catalog/ID for the viewer configuration.</p> <p>Specifies an image catalog entry that contains the viewer configuration properties in <code>catalog::UserData</code>. When this command is present, the viewer sends a <code>req=userdata</code> command for <code>configId</code> to the server and extracts properties from the reply. The properties are used to initialize the viewer. If the URL string specifies the same properties, they override the values from <code>catalog::UserData</code>.</p>
-----------------------	--

All viewer commands that can be specified in `catalog::UserData` expect `asset`, `serverUrl`, `contentUrl`, `searchServerUrl`, and `config` itself.

Properties

Optional.

Default

None.

Example 1

An image catalog named 2017 contains the entry `preset-oct`. The `catalog::UserData` field of this catalog entry includes the following data:

```
style=customStyle.css
```

Load the viewer with the following command:

```
config=2017/preset-oct
```

This is equivalent to the following commands specified explicitly in the URL:

```
style=customStyle.css
```

Example 2

An image catalog named 2017 contains the entry `spin-oct`. The `catalog::UserData` field of this catalog entry includes the following data:

```
zoomStep=3
maxZoom=200
```

Load the viewer with the following command:

```
config=2017/spin-oct
```

This is equivalent to the following commands specified explicitly in the URL:

```
zoomStep=3&maxZoom=200
```

Example 3

A viewer preset named `Shoppable_Banner` includes the following data:

```
style=etc/dam/presets/css/html5_interactiveimage.css
```

Load the viewer with the following command:

```
config=/etc/dam/presets/viewer/Shoppable_Banner
```

This is equivalent to the following commands specified explicitly in the URL:

```
style=etc/dam/presets/css/html5_interactiveimage.css
```

Example 4

A viewer preset named Shoppable_Video_Dark contains the following data:

```
style=etc/dam/presets/css/html5_interactivevideo_dark.css
```

Load the viewer with the following command:

```
config=/etc/dam/presets/viewer/Shoppable_Video_Dark
```

This is equivalent to the following commands specified explicitly in the URL:

```
style=etc/dam/presets/css/html5_interactivevideo_dark.css
```

Example 5

A viewer preset named Carousel_Dotted_light the following data:

```
style= etc/dam/presets/css/html5_carouselviewer_dotted_light.css
```

Load the viewer with the following command:

```
config=/etc/dam/presets/viewer/Carousel_Dotted_light
```

This is equivalent to the following commands specified explicitly in the URL:

```
style= etc/dam/presets/css/html5_carouselviewer_dotted_light.css
```

config2

Parameter common to all viewers.



Note: This command does not apply to Video Image Viewer.

```
config2=companypreset
```

<i>companypreset</i>	Name of the Adobe Analytics configuration preset.
----------------------	---

Properties

Optional.

Default

None.

Example

```
config2=companypreset
```

contentType

Parameter common to all viewers.

`contentType=contentTypePath`

<code>contentTypePath</code>	<p>Specifies the base path to custom CSS files, any closed captioning content, or navigation content.</p> <p>If the path does not have a leading / , it is relative to the location of the viewer HTML page. If the path has a leading / , it specifies an absolute path on the same server.</p> <p>Does not affect the loading of the default CSS file when you do not specify a style command.</p>
------------------------------	--

Properties

Optional.

Default

`/is/content/`

Examples

```
contentType=/skins/
```

```
contentType=http://aodmarketingna.assetsadobe.com/
```

```
contentType=https://demos-pub.assetsadobe.com/
```

initialFrame

Parameter common to all viewers.



Note: This command does not apply to Video Image Viewer.

`initialFrame=frameIdx[,pageIdx]`

<code>frameIdx</code>	Specifies a zero-based frame index that the viewer displays on load.
<code>pageIdx</code>	<p>A zero-based index of the page within the spread when the device is in portrait orientation. In a "left-to-right" environment 0 means "left page" and 1 means "right page". In "right-to-left" it is opposite: 0 means "right page" and 1 means "left page".</p> <p>If not specified, 0 is assumed by default. Ignored when device is in landscape orientation.</p>

Properties

Optional.

Default

No default.

Example

```
initialFrame=2
```

serverUrl

Parameter common to all viewers.

```
serverUrl=isRootPath
```

<i>isRootPath</i>	<p>Relative or absolute Image Serving root path.</p> <p>Specifies a relative or absolute path to Image Serving, from where the viewer retrieves images. If the path does not have a leading /, it is relative to the location of the viewer HTML page. If the path has a leading /, it specifies an absolute path on the same server.</p> <p>Use only an absolute path in case the Email share module is enabled in the viewer.</p>
-------------------	---

Properties

Optional. Not needed for standard SaaS (software as a service) usage.

Default

```
/is/image/
```

Examples

```
serverUrl=http://s7d1.scene7.com/is/image/
```

```
serverUrl=http://aodmarketingna.assetsadobe.com/is/image
```

```
serverUrl=https://adobedemo62-h.assetsadobe.com/is/image
```

videoServerUrl

Parameter common to all viewers.



Note: This command does not apply to Video Image Viewer.

```
videoServerUrl=videoRootPath
```

<i>videoRootPath</i>	<p>The video server root path. If no domain is specified, then the domain from which the page is served is applied instead. Standard URI path resolution applies.</p>
----------------------	---

Properties

Optional. Not needed for standard software as a service usage.

Default

/is/content/

Example

```
videoServerUrl=http://s7d1.scene7.com/is/content/
```


Keyboard accessibility and navigation

All features exposed in the Basic Zoom, Flyout, Inline Zoom, Mixed Media, Spin, Video, Zoom, Interactive Image, and Interactive Video viewer interface are keyboard accessible.

An end user can navigate between viewer user interface elements using **Tab** and **Shift+Tab** keystrokes. Using **Tab** advances input focus to the next user interface element in the tabbing order; using **Shift+Tab** brings input focus back to the previous user interface element. The focus traversal follows the natural user interface element location on the screen and moves in a left-to-right, then top-to-bottom order.

Depending on the operating system and web browser settings, the user interface element that has input focus may receive a visual focus indication. For example, the visual indicator can be a thin border rendered around the user interface element.

It is possible to disable or customize such focus highlight in the viewer CSS. In the table of contents, under a specific viewer name (for example, Basic Zoom), click **Customizing HTML5 viewer name Viewer > Focus highlight** (excludes eCatalog and eCatalog Search).

Keystrokes supported by individual viewer user interface elements are, in most cases, obvious and easy to discover.

Action	Keystroke
Activate button components	Space or Enter
Zoom in or out	+ or - , respectively
Zoom reset	Esc
Pan	Up, down, left, or right arrow
Spinning a 360 degree image	Use arrow keys when the image is in a reset state. Up or down arrow can be used only when working with multi-dimensional spin sets.
Product swatch selection	Up, down, left, or right arrow; Home or End
Product swatch activation	Space or Enter
Video, gradual rewind	Left or up arrow
Video, fast forward	Right or down arrow
Video, go to beginning or end	Home or End , respectively
Video, control volume level when focus is on slider	Up, down, left, or right arrow; Home or End
Carousel, change banner image in main view	Left or right arrow

Action	Keystroke
Carousel, hotspot selection and hotspot activation	Hotspot selection: up, down, left, or right arrow key Hotspot activation: space bar or Enter key.

Viewer SDK Tutorial

The Viewer SDK provides a set of JavaScript-based components for custom viewer development. The viewers are web-based applications that allow for rich media content served by Adobe Scene7 to be embedded in web pages.

For example, the SDK provides interactive zooming and panning. It also provides 360° view and video playback of assets that were uploaded to Adobe Scene7 through the backend application called SPS (Scene7 Publishing System).

Even though the components rely on HTML5 functionality, they are designed to work on Android and Apple iOS devices, and desktops, including Internet Explorer and later. This kind of experience means that you are able to provide a single workflow for all supported platforms.

The SDK consists of UI Components that make up viewer content. You can style these components through CSS, and non-UI components that have some kind of supporting role, like set definition fetching and parsing or tracking. All component behaviors are customizable through modifiers that you can specify in a number of ways, for example, as `name=value` pairs in the URL.

This tutorial includes the following order of tasks to help you create a basic zoom viewer:

- [Download the latest Viewer SDK from Adobe Developer Connection](#)
- [Load the Viewer SDK](#)
- [Adding style to your viewer](#)
- [Including Container and ZoomView](#)
- [Adding MediaSet and Swatches components to your viewer](#)
- [Adding buttons to your viewer](#)
- [Configuring the Swatches vertically](#)

Download the latest Viewer SDK from Adobe Developer Connection

1. Download the latest Viewer SDK from Adobe Developer Connection [here](#).



Note: You can complete this tutorial without the need to download the Viewer SDK package because the SDK is actually loaded remotely. However, the Viewer package includes additional examples and an API reference guide that you will find helpful when you create your own viewers.

Load the Viewer SDK

1. Start by setting up a fresh page to develop the basic zoom viewer that you are going to create.

Consider this the bootstrap—or loader—code to set up an empty SDK application. Open your favorite text editor and paste the following HTML markup into it:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="user-scalable=no, height=device-height,
width=device-width, initial-scale=1.0, maximum-scale=1.0"/>

    <!-- Hiding the Safari on iPhone OS UI components -->
    <meta name="apple-mobile-web-app-capable" content="yes"/>
    <meta name="apple-mobile-web-app-status-bar-style" content="black"/>
    <meta name="apple-touch-fullscreen" content="no"/>

    <title>Custom Viewer</title>
```

```

        <!--
            Include Utils.js before you use any of the SDK components. This file
            contains SDK utilities and global functions that are used to initialize the
viewer and load viewer
            components. The path to the Utils.js determines which version of the SDK that
the viewer uses. You
            can use a relative path if the viewer is deployed on one of the Adobe Scene7
servers and it is served
            from the same domain. Otherwise, specify a full path to one of Adobe Scene7
servers that have the SDK
            installed.
        -->
        <script language="javascript" type="text/javascript"
            src="http://s7dl.scene7.com/s7sdk/2.8/js/s7sdk/Utils.js"></script>

    </head>
    <body>
        <script language="javascript" type="text/javascript">
        </script>
    </body>
</html>

```

Add the following JavaScript code inside the script tag to initialize the ParameterManager. This helps you prepare to create and instantiate SDK components inside the initView function:

```

/* We create a self-running anonymous function to encapsulate variable scope. Placing code
inside such
    a function is optional, but this prevents variables from polluting the global object.
*/
(function () {

    // Initialize the SDK
    s7sdk.Util.init();

    /* Create an instance of the ParameterManager component to collect components'
configuration
    that can come from a viewer preset, URL, or the HTML page itself. The ParameterManager
    component also sends a notification s7sdk.Event.SDK_READY when all needed files are
loaded
    and the configuration parameters are processed. The other components should never be
initialized
    outside this handler. After defining the handler for the s7sdk.Event.SDK_READY event,
it
    is safe to initiate configuration initialization by calling ParameterManager.init().
*/
    var params = new s7sdk.ParameterManager();

    /* Event handler for s7sdk.Event.SDK_READY dispatched by ParameterManager to initialize
various components of
    this viewer. */
    function initView() {

    }

    /* Add event handler for the s7sdk.Event.SDK_READY event dispatched by the ParameterManager
when all modifiers
    are processed and it is safe to initialize the viewer. */
    params.addEventListener(s7sdk.Event.SDK_READY, initView, false);

    /* Initiate configuration initialization of ParameterManager. */
    params.init();

})();

```

2. Save the file as an empty template. You can use any filename you want.

You will use this empty template file as a reference when creating any new viewers in the future. This template works locally and when served from a web server.

You will now add style to your viewer.

Adding style to your viewer

1. For this full page viewer that you are creating, you can add some basic styles.

Add the following style block to the bottom of the head:

```
<style>
  html, body {
    width: 100%;
    height: 100%;
  }
  body {
    /* Remove any padding and margin around the edges of the browser window */
    padding: 0;
    margin: 0;

    /* We set overflow to hidden so that scroll bars do not flicker when resizing the
window */
    overflow: hidden;
  }
</style>
```

You will now include the components Container and ZoomView.

Including Container and ZoomView

1. Create an actual viewer by including the components Container and ZoomView.

Insert the following include statements to the bottom of the <head> element—after the Utils.js script is loaded:

```
<!--
  Add an "include" statement with a related module for each component that is needed for
that particular
  viewer. Check API documentation to see a complete list of components and their modules.
-->
<script language="javascript" type="text/javascript">
  s7sdk.Util.lib.include('s7sdk.common.Container');
  s7sdk.Util.lib.include('s7sdk.image.ZoomView');
</script>
```

2. Now create variables to reference the various SDK components.

Add the following variables to the top of the main anonymous function, just above s7sdk.Util.init():

```
var container, zoomView;
```

3. Insert the following inside the initView function to define some modifiers and instantiate the respective components:

```
/* Modifiers can be added directly to ParameterManager instance */
params.push("serverurl", "http://s7dl.scene7.com/is/image");
params.push("asset", "Scene7SharedAssets/ImageSet-Views-Sample");

/* Create a viewer container as a parent component for other user interface components that
are part of the viewer application and associate event handlers for resize and
full screen notification. The advantage of using Container as the parent is the
component's ability to resize and bring itself and its children to full screen. */
container = new s7sdk.common.Container(null, params, "s7container");
container.addEventListener(s7sdk.event.ResizeEvent.COMPONENT_RESIZE, containerResize, false);
```

```
/* Create ZoomView component */
zoomView = new s7sdk.image.ZoomView("s7container", params, "myZoomView");

/* We call this to ensure all SDK components are scaled to initial conditions when viewer
loads */
resizeViewer(container.getWidth(), container.getHeight());
```

4. For the above code to run properly, add a containerResize event handler and a helper function:

```
/* Event handler for s7sdk.event.ResizeEvent.COMPONENT_RESIZE events dispatched by Container
to resize
various view components included in this viewer. */
function containerResize(event) {
    resizeViewer(event.s7event.w, event.s7event.h);
}

/* Resize viewer components */
function resizeViewer(width, height) {
    zoomView.resize(width, height);
}
```

5. Preview the page so you can see what you have created. Your page will look like the following:



You will now add the components `MediaSet` and `Swatches` to your viewer.

Adding `MediaSet` and `Swatches` components to your viewer

1. To give users the ability to select images from a set, you can add the components `MediaSet` and `Swatches`.

Add the following SDK includes:

```
s7sdk.Util.lib.include('s7sdk.set.MediaSet');
s7sdk.Util.lib.include('s7sdk.set.Swatches');
```

2. Update the variable list with the following:

```
var mediaSet, container, zoomView, swatches;
```

3. Instantiate `MediaSet` and `Swatches` components inside the `initViewer` function.

Be sure to instantiate the `Swatches` instance after the `ZoomView` and `Container` components, otherwise the stacking order hides the `Swatches`:

```
// Create MediaSet to manage assets and add event listener to the NOTF_SET_PARSED event
mediaSet = new s7sdk.set.MediaSet(null, params, "mediaSet");

// Add MediaSet event listener
mediaSet.addEventListener(s7sdk.event.AssetEvent.NOTF_SET_PARSED, onSetParsed, false);

/* create Swatches component and associate event handler for swatch selection notification */
swatches = new s7sdk.set.Swatches("s7container", params, "mySwatches");
swatches.addEventListener(s7sdk.event.AssetEvent.SWATCH_SELECTED_EVENT, swatchSelected, false);
```

4. Now add the following event handler functions:

```
/* Event handler for the s7sdk.event.AssetEvent.NOTF_SET_PARSED event dispatched by MediaSet
to
    assign the asset to the Swatches when parsing is complete. */
function onSetParsed(e) {

    // set media set for Swatches to display
    var mediasetDesc = e.s7event.asset;
    swatches.setMediaSet(mediasetDesc);

    // select the first swatch by default
    swatches.selectSwatch(0, true);
}

/* Event handler for s7sdk.event.AssetEvent.SWATCH_SELECTED_EVENT events dispatched by
Swatches to switch
    the image in the ZoomView when a different swatch is selected. */
function swatchSelected(event) {
    zoomView.setItem(event.s7event.asset);
}
```

5. Position the swatches at the bottom of the viewer by adding the following CSS to the style element:

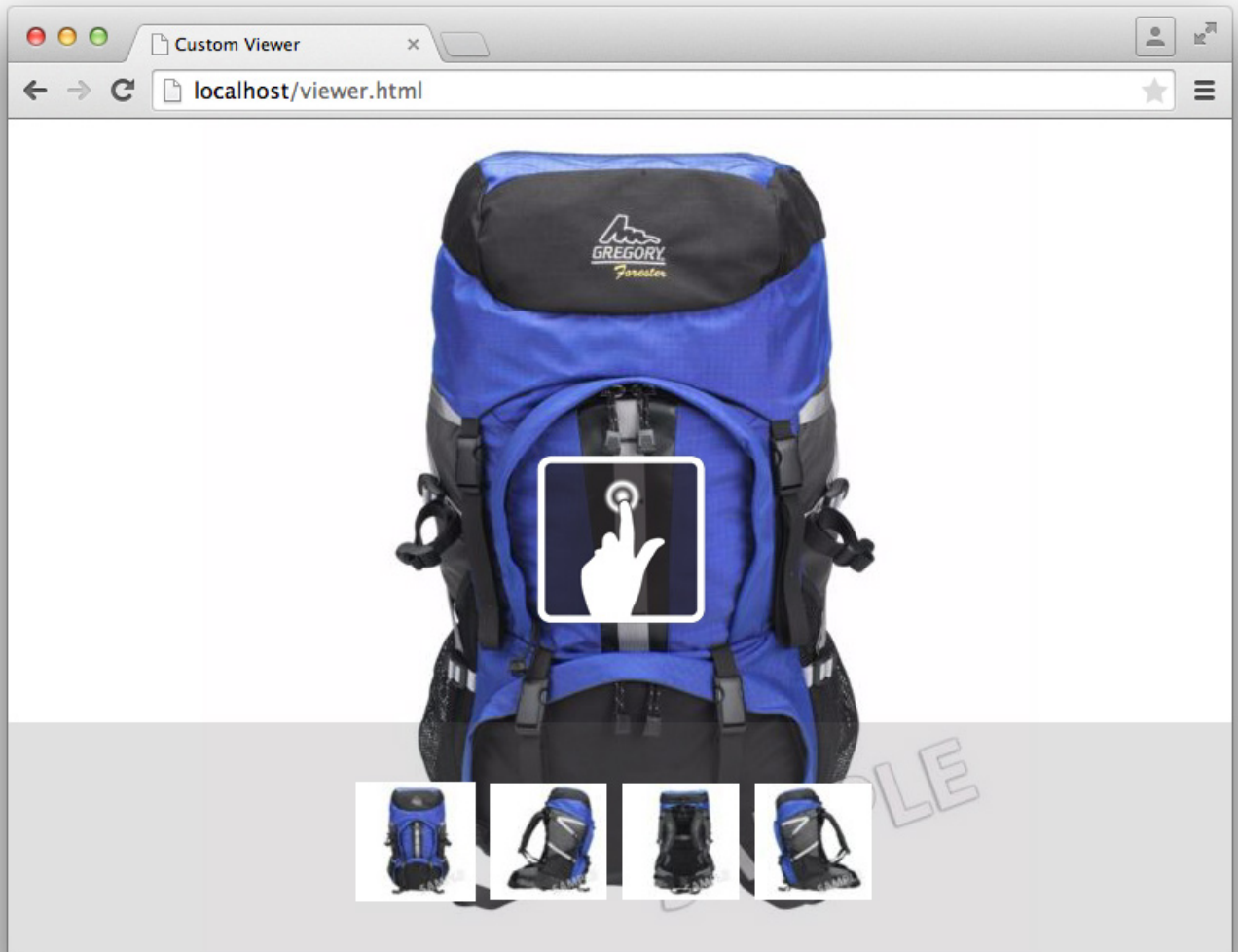
```
/* Align swatches to bottom of viewer */
.s7swatches {
    bottom: 0;
    left: 0;
    right: 0;
    height: 150px;
}
```

6. Preview your viewer.

Notice that the swatches are in the lower left of the viewer. To have the swatches to take the entire viewer width, add a call to manually resize the swatches whenever the user resizes their browser. Add the following to the `resizeViewer` function:

```
swatches.resize(width, swatches.getHeight());
```

Your viewer now looks like the following image. Try resizing the browser window of the viewer and notice the resulting behavior.



You will now add zoom in, zoom out, and zoom reset buttons to your viewer.

Adding buttons to your viewer

1. Currently, the user can only zoom using click or touch gestures. Therefore, add some basic zoom control buttons to the viewer.

Add the following button components:

```
s7sdk.Util.lib.include('s7sdk.common.Button');
```


2. Update the variable list with the following:

```
var mediaSet, container, zoomView, swatches, zoomInButton, zoomOutButton, zoomResetButton;
```

3. Instantiate Buttons at the bottom of `initViewer` function.

Remember that the order matters, unless you specify the `z-index` in CSS:

```
/* Create Zoom In, Zoom Out and Zoom Reset buttons */
zoomInButton = new s7sdk.common.ZoomInButton("s7container", params, "zoomInBtn");
zoomOutButton = new s7sdk.common.ZoomOutButton("s7container", params, "zoomOutBtn");
zoomResetButton = new s7sdk.common.ZoomResetButton("s7container", params, "zoomResetBtn");

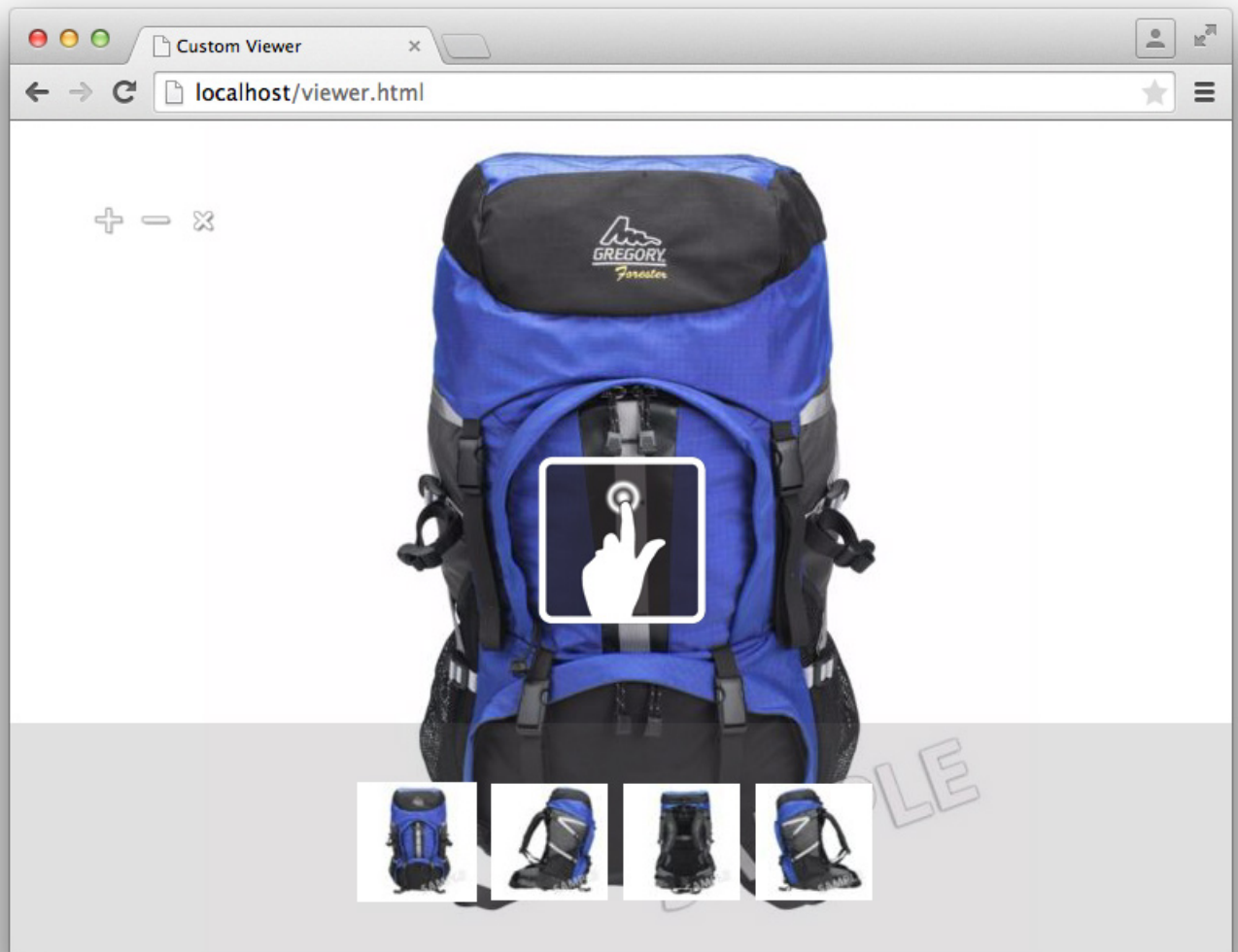
/* Add handlers for zoom in, zoom out and zoom reset buttons inline. */
zoomInButton.addEventListener("click", function() { zoomView.zoomIn(); });
zoomOutButton.addEventListener("click", function() { zoomView.zoomOut(); });
zoomResetButton.addEventListener("click", function() { zoomView.zoomReset(); });
```

4. Now define some basic styles for the buttons by adding the following to the style block at the top of your file:

```
/* define styles common to all button components and their sub-classes */
.s7button {
    position: absolute;
    width: 28px;
    height: 28px;
    z-index: 100;
}

/* position individual buttons*/
.s7zoominbutton {
    top: 50px;
    left: 50px;
}
.s7zoomoutbutton {
    top: 50px;
    left: 80px;
}
.s7zoomresetbutton {
    top: 50px;
    left: 110px;
}
```

5. Preview your viewer. It will look like the following:



You will now configure the Swatches so that they are aligned vertically on the right.

Configuring the Swatches vertically

1. You can configure modifiers directly on the `ParameterManager` instance.

Add the following to the top of the `initViewer` function to configure the Swatches thumb layout as a single row:

```
params.push("Swatches.tmblayout", "1,0");
```

2. Update the following `resize` call inside `resizeViewer`:

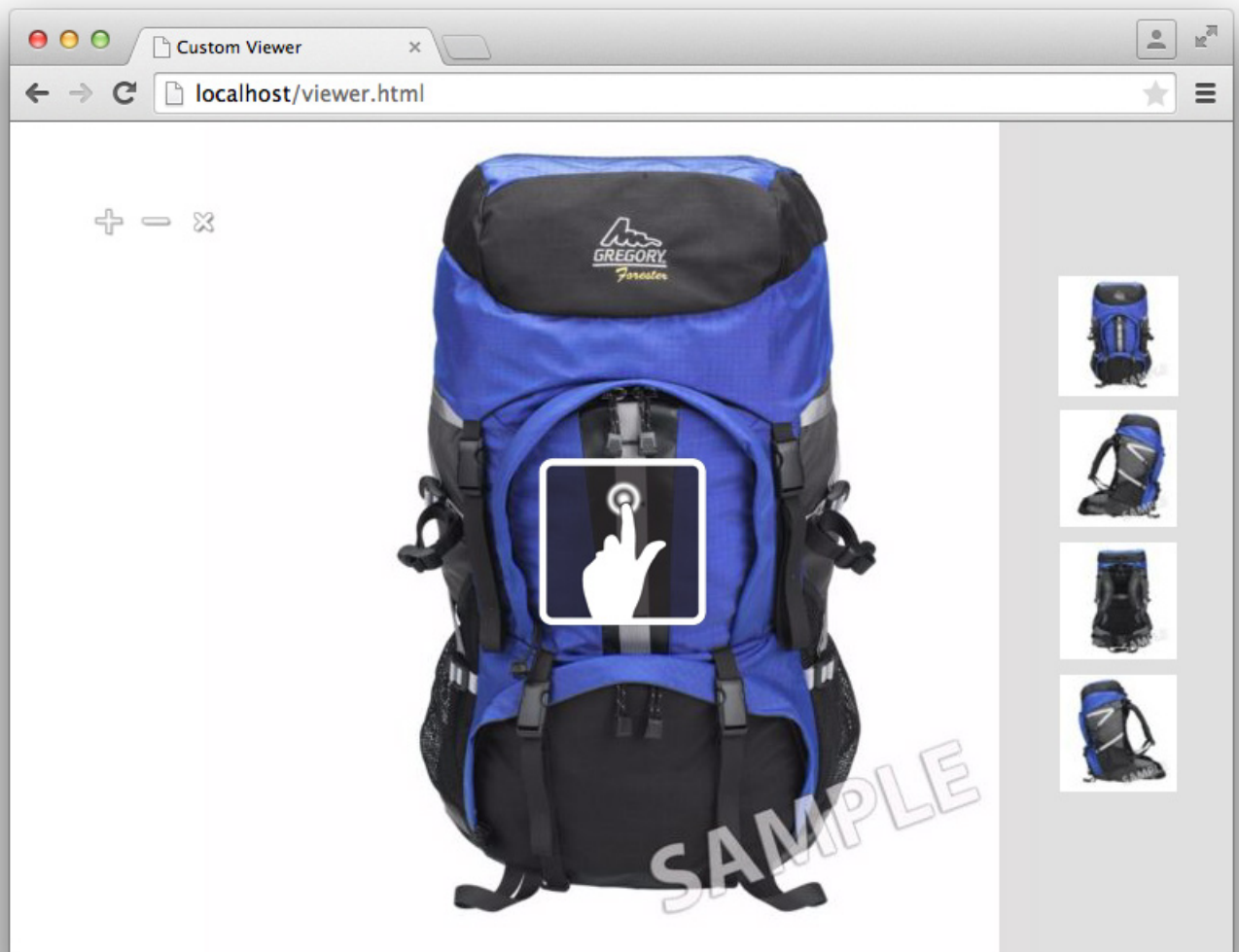
```
swatches.resize(swatches.getWidth(), height);
```

3. Edit the following `s7swatches` rule in `ZoomViewer.css`:

```
.s7swatches {  
  top:0 ;  
  bottom: 0;  
  right: 0;
```

```
width: 150px;  
}
```

4. Preview your viewer. It will look like the following:



Your basic zoom viewer is now complete.

This viewer tutorial touches on the fundamentals of what the Scene7 Viewer SDK provides. As you work with the SDK you can use the various standard components to easily build and style rich viewing experiences for your target audiences.