

PanoramaStudio Viewer 4.77

Interactive Panoramas, Virtual Tours and Zoom Images on webpages with
HTML5

Documentation

© 2005-2024, Tobias Huellmandel Software
<https://www.tshsoft.com>

Contents

1	The PanoramaStudio Viewer	1
1.1	Overview and system requirements	1
1.2	Features	1
1.2.1	Overview	1
1.2.2	Supported browsers	1
1.3	Using the Viewer with the PanoramaStudio application	2
1.4	Using the Viewer with panoramic images created with other software	2
1.5	Controlling the PanoramaStudio Viewer	2
1.6	Compatibility with old XML parameter files	2
1.7	License	3
2	Creating the image data for the Viewer	6
3	Embedding the Viewer into a webpage	6
3.1	Embedding the Viewer directly into HTML	6
3.2	Embed the Viewer with an <iframe>	7
3.3	Add and remove the Viewer dynamically	7
3.4	Required files for the Viewer	8
4	Parameters Reference	9
4.1	Overview	9
4.2	Objects	10
	actions	10
	audio	10
	autoplay	11
	button	11
	buttonBox	12
	camera	13
	contextmenu	13
	control	14
	data	15
	events	15
	gallery	15
	galleryElement	16

gfx	17
hotspot	17
hotspots	18
image	19
include	20
key	20
lensflare	21
location	21
logo	21
mapElement	22
node	23
orientation	23
settings	24
textbox	24
toolbar	25
translate	25
video	26
view	26
4.3 All-purpose properties	27
4.3.1 Properties of all objects	27
4.3.2 Properties of all layers	27
4.3.3 Properties of Button and Orientation	28
5 JavaScript API	29
5.1 Access a panorama object	29
5.2 Events	29
5.2.1 Overview	29
5.2.2 Events of a panoStudioViewer instance	29
5.2.3 Local events on each layer	31
5.3 Functions	33
5.3.1 Functions of the global, static panoStudioViewer object	33
5.3.2 Functions of the Viewer instance	33
5.3.3 Local functions of layers	40
5.3.4 Local functions of buttons	42
5.3.5 Local functions of textboxes	42
5.3.6 Local functions of audio objects	43
5.4 Properties	43

1 The PanoramaStudio Viewer

1.1 Overview and system requirements

The *PanoramaStudio Viewer* provides an interactive player for displaying 3D panoramas, virtual tours and large zoom images on webpages. The *PanoramaStudio Viewer* is a small and highly optimized HTML5 application, which runs in any HTML5 compatible webbrowser.

With this requirements the Viewer is able to show nearly arbitrarily large images and panoramas embedded in webpages in a platform-independent way on a wide range of systems. With the ability to connect panoramas using hotspots it is also possible to build virtual tours containing multiple panoramas.

1.2 Features

1.2.1 Overview

- Interactive and platform-independent presentation of panoramas and large images on a local computer as well as online
- Small, highly-performant HTML5 viewer in a single JavaScript file
- Supports partial panoramas, full spherical 360x180 degree panoramas, gigapixel panoramas, virtual tours and 2D zoom images.
- Fast and performant displaying of nearly arbitrarily large images using dynamic loading
- Highly optimized JavaScript code
- Supports spherical, cylindrical and cubic panorama formats
- VR mode via WebXR, WebVR or emulated through split-screen display
- JavaScript interface to control the Viewer
- Supports hotspots and virtual tours
- Supports simulation of artificial lens flares and allows embedding audio files for a more realistic environment
- Auto Play for automatic panning and zooming in the panorama
- Supports motion sensors
- Extensive configuration options

1.2.2 Supported browsers

The current version of the PanoramaStudio Viewer runs in the following HTML5 compatible browsers:

- Safari iOS 4+
- Browser on Android 4.1+
- Opera 15+ (desktop)
- Opera 14+ (mobile)
- Chrome 20+ (desktop)
- Firefox 18+ (desktop)
- Firefox 18+ (mobile)
- Safari 5+ (desktop)
- Internet Explorer 10+

1.3 Using the Viewer with the PanoramaStudio application

The *PanoramaStudio* is an integral part of the PanoramaStudio application. Webpage templates with an embedded interactive panorama or a zoom image can be created automatically and easily using PanoramaStudio. PanoramaStudio generates fully automated the necessary HTML code and the necessary files according to the user's settings for this purpose. The PanoramaStudio export supports already many of the parameters and options of the Viewer described in this documentation.

Notice on how to create a panorama for the Viewer

For the creation of an interactive panorama or zoom image you should first of all use the export functionality in the PanoramaStudio application. This creates an initial version of the panorama fully automated.

Since all the features and possibilities of the Viewer described in this documentation are going far beyond the configuration options in the PanoramaStudio application, this documentation allows advanced users to edit and modify every aspect and detail in an existing Viewer panorama created with PanoramaStudio.

All features and parameters which are available for an individual adjustment of the Viewer are described in the following.

1.4 Using the Viewer with panoramic images created with other software

As a matter of principle the Viewer handles and shows all cylindrical and spherical panoramas and also flat images created with other software. Therefore the images can be imported in the PanoramaStudio application using *File→Import/Export→Import panoramic image...* and then converted and exported for the Viewer.

1.5 Controlling the PanoramaStudio Viewer

Inside a panorama showed with the *PanoramaStudio Viewer* the user can navigate interactively. You can pan horizontally and tilt vertically inside the scene using mouse or touch input. Additionally the mouse wheel is supported to zoom in and out. If the toolbar is shown in the Viewer, there are furthermore buttons for the navigation, for starting and stopping the Auto Play, and other functions.

1.6 Compatibility with old XML parameter files

The PanoramaStudio Viewer uses JavaScript/JSON as format for parameter files since version 4.0. It still reads the previously used XML based parameter files however.

But some browsers (e.g. chrome) block reading XML files with JavaScript from the local filesystem, such that parameter files can be read by this browsers only by *http://* and *https://*.

1.7 License

License agreement

PanoramaStudio Viewer - Version 4
for HTML5 compatible browsers
Copyright © 2005-2022, Tobias Huellmandel Software
Internet: <https://www.tshsoft.com>
E-Mail: support@tshsoft.com

LICENSE AGREEMENT

1. LICENSE: The author (Tobias Huellmandel Software) provides you the software "PanoramaStudio Viewer 4", named "Software" in the following, which contains further an electronic documentation and a license file ("License.txt") and grants you a license for using this product in accordance with the following conditions.
2. COMMERCIAL/PRIVATE USE:
 - 2.1 PRIVATE USE
The private/non-commercial use of the Software, also on the internet, is free of charge and requires no further license. In addition, non-profit organizations and academic institutions also do not require an additional license.

2.2 COMMERCIAL USE

If the software is used commercially on the Internet, a commercial license must be obtained after a 30-day trial at the latest. A commercial use on the Internet exists in principle, if the operator of the website uses this commercially, is not a private person, not a non-profit organization or an academic institution.

The following licenses are available:

SINGLE-DOMAIN-LICENSE

A Single-Domain-License is valid for one website and its domain, as well as any existing alias domain names that refer to the same website.

MULTI-DOMAIN-LICENSE

A Multi-Domain-License may be used on any number of websites, including third-party websites.

The number of panoramas displayed on a web page with the licensed software is not limited.

3. LICENSE RESTRICTIONS:

- 3.1 You may not use this Software in a way other than allowed in this License.
- 3.2 You may not rent, lease or sublicense the Software without prior written permission.
- 3.3 You shall not use an UNREGISTERED copy of the Software on a commercial website after the end of a 30-day trial period.
- 3.4 Under the above licenses, it is not permitted to use the software for a hosting platform that allows third parties to create content displayed with the Software.
If you are interested in a hosting license please contact support@tshsoft.com

- 3.5 You may not alter, modify, adapt or translate the Software, or decompile, reverse engineer, or disassemble the Software.
- 3.6 You may not modify the Software or create derivative works based upon the Software.

4. THE SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING ANY WARRANTY OF QUALITY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL THE AUTHOR BE LIABLE FOR LOSS OF DATA, COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES OR ANY SPECIAL, CONSEQUENTIAL OR INCIDENTAL DAMAGES, UNDER ANY CAUSE OF ACTION AND REGARDLESS OF WHETHER OR NOT THE AUTHOR HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THIS LIMITATION WILL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY PROVIDED HEREIN. IN ANY EVENT THE AUTHOR WILL HAVE LIABILITY ARISING OUT OF THIS AGREEMENT.

Parts of the software are based on Robert Penner's Easing Functions:

TERMS OF USE - EASING EQUATIONS

Open source under the BSD License.

Copyright © 2001 Robert Penner
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
Neither the name of the author nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Purchase a License

According to the above license agreement a license key is needed for the commercial use of the *PanoramaStudio Viewer* on the internet after a 30-day trial period. For such a commercial use you can buy your license at the registration service Share-It!. With the purchase you will get a license key compatible to the domain name of your internet presence.

With the license key you will get:

- The right to use the *PanoramaStudio Viewer* on commercial websites. The number of panoramas showed with a licensed copy of the Viewer on a website is not limited.
- The option to show a custom logo in the Viewer.
- Free support.

Order a license key

A license key can be ordered easily and primarily secure over the internet at the registration service Share-It!. We accept credit cards, checks, bank/wire transfer, or cash as payment method. The delivery of your license key for a certain domain name is done by e-mail within 24 hours after receiving your payment. If you pay by credit card the license key is delivered in most cases within 30 to 60 seconds. Order form: https://www.tshsoft.com/en/order_en

Customer service

Do you have questions about your order, payment or the delivery? Do you have already ordered a product and want to check your data? You will find the answers at the Customer Care Center of Share-It!: <https://secure.shareit.com/shareit/ccc/index.html?publisherid=20959&languageid=1>

2 Creating the image data for the Viewer

The *PanoramaStudio Viewer* is able to show images of nearly arbitrary size. Therefore a conversion of the image data into an optimized format is necessary, where the original image data is split up into small tiles of different resolutions. This enables the Viewer to dynamically load and display the required image regions for the current field-of-view. This conversion of the image data for the Viewer requires the PanoramaStudio application. Existing (panoramic) images can be imported into PanoramaStudio using the command *File→Import/Export→Import panoramic image....* PanoramaStudio automatically creates optimized image data for the Viewer when saving an interactive panorama.

3 Embedding the Viewer into a webpage

3.1 Embedding the Viewer directly into HTML

To embed the PanoramaStudio Viewer directly into HTML you have to load the JavaScript of the Viewer (`panoStudioViewer.js`) in the page.

Then call the function `panoStudioViewer.insert(...)` to assign the Viewer to a DIV in the page, load the panorama through its parameter file and set optionally more parameters:

```
panoStudioViewer.insert(id, parameterFile, inlineParameters, startupCallback)
```

This function is documented in chap. 5.3.1 (→ p.33).

Example:

The following HTML code will be created in similar form automatically by PanoramaStudio when you save an interactive panorama. Hence you can just copy and paste this snippet to embed a panorama into another webpage.

You find a detailed description of the structure of the parameter file (`MyPano.js` here) in chap. 4 (→ p.9).

The id `viewerID` in the div element also allows to control the panorama using JavaScript (chap. 5, p.29).

```
<div style="margin: 0 auto; width: 1024px; height: 540px; text-align: center;">
  <noscript><div style="color: red; width: 30%; border: 1px solid red; padding: 4px;
    font-family: sans-serif;">ERROR: Your web browser must have JavaScript enabled to
    show this panorama.</div></noscript>

  <div id="viewerID" style="position: relative; margin: auto; padding: 0px; left: 0px;
    top: 0px; width: 100%; height: 100%;"></div>

  <script src="panoStudioViewer.js"></script>
  <script>
    panoStudioViewer.insert("viewerID", "MyPano.js", {html5: "disable_webgl_warning"});
  </script>
</div>
```

The same example with the size of the panorama set to 100% to fill the browser window:

```
<div style="margin: 0 auto; height: 100%; overflow: hidden; text-align: center;">
  <noscript><div style="color: red; width: 30%; border: 1px solid red; padding: 4px;
    font-family: sans-serif;">ERROR: Your web browser must have JavaScript enabled to
    show this panorama.</div></noscript>

  <div id="viewerID" style="position: absolute; left: 0px; top: 0px; width: 100%;
    height: 100%"></div>

  <script src="panoStudioViewer.js"></script>
  <script>
    panoStudioViewer.insert("viewerID", "MyPano.js", {html5: "disable_webgl_warning"});
  </script>
</div>
```

3.2 Embed the Viewer with an <iframe>

You can also simply embed a panorama HTML created with PanoramaStudio into any other HTML page with an <iframe> element:

```
<iframe src="/path/to/the/panorama.html" width="[as desired]" height="[as desired]"
  scrolling="no" marginheight="0" marginwidth="0" frameborder="0"
  allowfullscreen mozallowfullscreen webkitallowfullscreen> </iframe>
```

If you save the interactive panorama with the option *Fit to page*, it will adjust its size according to the size of the iframe element.

Example:

```
<iframe src="MyPanorama.html" width="1024px" height="480px"
  scrolling="no" marginheight="0" marginwidth="0" frameborder="0"
  allowfullscreen mozallowfullscreen webkitallowfullscreen> </iframe>
```

3.3 Add and remove the Viewer dynamically

Viewer objects can be added and removed to a HTML also dynamically on user interactions similar to chap. 3.1 (→ p.6).

Therefore the Viewer JavaScript must be included in the HTML. For example:

```
<script src="panoStudioViewer.js"></script>
```

Furthermore a HTML DIV element must be present as container for the Viewer object. For example:

```
<div id="panoramaContainer" style="width: 600px; height: 400px;"></div>
```

You can add a panorama object dynamically on user interaction. For example:

```
<input type="button" value="Add panorama"
  onclick="panoStudioViewer.insert('panoramaContainer', 'panorama.js');" />
```

The panorama can be removed similarly:

```
<input type="button" value="Remove panorama"
  onclick="panoStudioViewer.remove('panoramaContainer');" />
```

3.4 Required files for the Viewer

When you save an interactive panorama / zoom image in PanoramaStudio then it creates already all necessary files. If you copy a panorama or upload it to a webserver ensure that you copy all of these files. A webpage with an interactive panorama or zoom image involves, besides the HTML file, several files. These are the PanoramaStudio Viewer as JavaScript file `panoStudioViewer.js`, a folder containing the image data and one or more parameter files (`.js` or `.json`) containing all the settings and parameters of the panorama.

These files should remain together in the same folder wherever applicable. Otherwise you have to adjust the paths in the HTML and parameter files.

4 Parameters Reference

4.1 Overview

All parameters for describing the properties and the display of the panorama are summarized in a parameter file, which basically consists of a JSON object (*JavaScript Object Notation*). See also <https://en.wikipedia.org/wiki/JSON>.

The structure of the parameter file consists of an assignment of the main JSON object to the global variable `PanoramaStudioViewerParams`. This object must contain an object `panoStudioViewer` which contains all objects for describing the panorama.

For a better overview, PanoramaStudio can output the file as a .json file. Strictly speaking, however, because of the assignment, it is JavaScript. For better compatibility on various, current web servers, the parameter files should therefore be created as .js.

Parameter files can contain functions ([Function] in the following). According to the declared data type of the parameter file, JSON or JS, functions must be embedded either as string (JSON) or can be embedded directly as JavaScript.

In order for the property to comply with the JSON format, the function must be a string in the following form:

```
"function(){ /* insert your code here */ }"
```

If the parameter files are declared as JavaScript (.js), functions can be embedded directly instead:

```
function(){ /* insert your code here */ }
```

Supported objects:

```
PanoramaStudioViewerParams = {
  "panoramaStudioViewer": {

    "actions": [...],
    "audio": [...],
    "autoplay": [...],
    "button": [...],
    "buttonBox": [...],
    "camera": [...],
    "contextmenu": [...],
    "control": [...],
    "data": [...],
    "events": [...],
    "gallery": [...],
    "galleryElement": [...],
    "gfx": [...],
    "hotspot": [...],
    "hotspots": [...],
    "image": [...],
    "include": [...],
    "key": [...],
    "lensflare": [...],
    "location": [...],
    "logo": [...],
```

```

    "mapElement": [...],
    "node": [...],
    "orientation": [...],
    "textbox": [...],
    "toolbar": [...],
    "translate": [...],
    "settings": [...],
    "video": [...],
    "view": [...]
  }
}

```

4.2 Objects

"actions":

Array of objects. Each object contains a function (`func`) and an ID (`id`). Used to store functions used by multiple objects. Functions can be called with the `action()` function (chap. 5.3.2 (→ p.35)).

Properties of each array item:

- `func` - A function with an arbitrary number of arguments.
- `viewer` - [Object], A variable to access the Viewer instance (see chap. 5.1 (→ p.29))
- properties inherited from object: chap. 4.3.1 (→ p.27)
- each array element provides the functions `get()` and `tween()` as defined in chap. 5.3.3 (→ p.40).

Example:

```

"actions": [
  {
    "func": function(txt){ console.log(txt); },
    "id": "printToConsole"
  }
]

```

"audio":

Embed an audio file.

Properties:

- `src` - [String], URL of a MP3 file
- `src_alt` - [String], Optional URL of an OGG audio file, as alternative audio source for browsers which do not support MP3.
- `loop` - [Boolean], If `true`, the audio will start over again, every time it is finished (default: `true`).
- `autoplay` - [Boolean], Plays the audio file automatically as soon as it is loaded when set to `true` (default: `true`).

- **volume** - [Number], Volume (default: 1.0, min: 0.0, max: 1.0).
- **priority** - [Number], Priority of the audio element. If there is a global and a local audio element, the element with the higher priority is played; if the priority is the same, the local element is played (default: 0)
- Properties inherited from object: chap. 4.3.1 (→ p.27)

Example:

```
"audio": {  
  "src": "music.mp3",  
  "loop": true,  
  "autoplay": true  
}
```

"autoplay":

Parameters regarding the Auto-Play.

Properties:

- **status** - [String], Possible values: 'on' or 'off' to switch the Auto-Play on or off.
- **reference** - [String], Defines the unit for the values in the attributes **pan** and **tilt**. Allowed values are 'pano' and 'screen'. With 'pano' pan and tilt move the image in degrees per second, with 'screen' the image is moved as a proportion of the screen width per second.
- **pan** - [Number], Horizontal panning speed of 3D a panorama (values <0 to the left, values >0 to the right). Unit defined in 'reference'.
- **tilt** - [Number], Vertical panning speed of 3D a panorama (values <0 to the top, values >0 to the bottom). Unit defined in 'reference'.
- **x** - [Number], Horizontal scrolling speed of a zoom image in parts of the image width per second (values <0 to the left, values >0 to the right).
- **y** - [Number], Vertical scrolling speed of a zoom image in parts of the image height per second (values <0 to the top, values >0 to the bottom).
- **zoom** - [Number], Zoom in or zoom out. Values less than 1.0 perform a zoom in, values bigger than 1.0 perform a zoom out.
- **restart** - Restart of the Auto-Play after an interruption due to an user interaction in seconds (default: 5.0, min: 0, max: 1000)
- **returnToLevel** - [Boolean], Slowly tilts back to the original level during horizontal auto-play if the user has previously tilted up or down (default: true)

"button":

Programmable button for navigation and other purposes.

Properties:

- **width** - [Number], Width in pixels.
- **height** - [Number], Height in pixels.
- **type** - [Number], Some predefined actions when the button is pressed (default: 0, no predefined action). Possible values:
 - 1: pan left
 - 2: pan right
 - 3: tilt up
 - 4: tilt down
 - 5: zoom in
 - 6: zoom out
- **priority** - [Number], Used for a *responsive* design, if the button is part of a `buttonBox`. If there is not enough space for all buttons of the `buttonBox` on a small display, then buttons with low priority will be hidden (e.g. the arrow buttons). Lowest priority is 0, the highest priority used by the PanoramaStudio export is 3.
- **index** - [Number], Used for the layout, if the button is part of a `buttonBox`. Buttons are arranged based on `index` in increasing order from left to right.
- **style** - [String], Equals the CSS *style* attribute in HTML. The object can be styled this way with CSS.
- **stylehover** - [String], CSS which is applied when the mouse hovers over the button. Overrides CSS properties possibly already specified in the `style` property.
- **styleactive** - [String], CSS which is applied when the mouse is pressed on the button. Overrides CSS properties possibly already specified in the `style` and `stylehover` properties.
- **skin** - [String], Describes how the button is painted. See chap. 4.3.3 (→ p.28).
- **skinhover** - [String], Specifies how the button is painted when the mouse cursor hovers over the button. See `skin`.
- **skinactive** - [String], Specifies how the button is painted when the mouse is pressed on a button. See `skin`.
- Properties inherited from `layer`: chap. 4.3.2 (→ p.27)
- Events inherited from `layer`: chap. 5.2.3 (→ p.31)

"buttonBox":

Represents a container for buttons. Amongst other things, you can specify several `button` properties, which will be applied on all buttons contained herein.

Properties:

- **width** - [String], Width in percent (%) or pixels (px). Example:

```
"width": "50%"
```

```
"width": "500px"
```

- **height** - [String], Height in percent (%) or pixels (px).
- **buttonWidth** - [Number], Width of the contained buttons in pixels. Each button can override this individually, specifying the width.
- **buttonHeight** - [Number], Height of the contained buttons in pixels.
- **spacing** - [Number], Spacing of the contained buttons in pixels (default: 8).
- **style** - [String], Equals the CSS *style* attribute in HTML. The object can be styled this way with CSS.
- **stylehover** - [String], CSS which will be applied when the mouse hovers over the `buttonBox`. Overrides CSS properties possibly already specified in the `style` property.
- **styleactive** - [String], CSS which will be applied when the mouse is pressed on the `buttonBox`. Overrides CSS properties possibly already specified in the `style` and `stylehover` properties.
- **buttonstyle** - [String], `style` property, which will be applied on all buttons contained in this `buttonBox`. See `button`.
- **buttonstylehover** - [String], `stylehover` property, which will be applied on all buttons contained in this `buttonBox`.
- **buttonstyleactive** - [String], `styleactive` property, which will be applied on all buttons contained in this `buttonBox`.
- **button** - [Array], Array of `button` objects, which are contained in this `buttonBox`. See chap. 4.2 (→ p.11).
If an entry contains the attribute `"type": "compass"` or `"type": "radar"`, then an orientation element is inserted in the button box. See also chap. 4.2 (→ p.23).
- Properties inherited from layer: chap. 4.3.2 (→ p.27)

"camera":

Camera properties. Limits the minimum and maximum zoom allowed. Defines for partial panoramas the covered field-of-view.

Properties:

- **minpan** - [Number], Left boundary of the panoramic image in degrees (default: 0)
- **maxpan** - [Number], Right boundary of the panoramic image (default: 360)
- **mintilt** - [Number], Upper boundary of the panoramic image (default: -90)
- **maxtilt** - [Number], Lower boundary of the panoramic image (default: 90)
- **minhfov** - [Number], Minimum horizontal field-of-view. This limits the maximum magnification (default: 0.000000001)
- **maxhfov** - [Number], Maximum horizontal field-of-view (default: 140)
- **maxzoom** - [Number], Maximum possible magnification of the image pixels (alternative to `minhfov`) (default: 10)

"contextmenu":

Configurable context menu.

Properties and functions:

- **highlightColor** - [String] Background color of the active menu item. Color specified in CSS format, e.g. blue, #ff0000 or rgb(255,0,0) .
- **highlightTextColor** - [String] Text color of the active menu item. Color specified in CSS format, e.g. blue, #ff0000 or rgb(255,0,0) .
- **style** - Equals the CSS *style* attribute in HTML. The context menu can be styled this way with CSS.
- **viewer** - [Object], A variable to access the Viewer instance (see chap. 5.1 (→ p.29)).
- **items** - [Array] Array of the menu items. See below the properties of the items.
- **update()** - [Function] Update the visibility and captions of the items in the context menu.
- **/*Object*/ getItem(/*String*/ id)** - [Function] Find a menu item in the items array based on its id.
- **/*Object*/ get(/*String*/ id)** - [Function] Find a layer or object based on its id.
- Events inherited from layer: chap. 5.2.3 (→ p.31)

Properties of the context menu items:

- **caption** - [String], Caption of the item.
- **id** - [String], ID of the item.
- **visible** - [Boolean], Visibility of the item.
- **viewer** - [Object], A variable to access the Viewer instance (see chap. 5.1 (→ p.29)).
- **oninit()** - [Function], Called when the item is initialized.
- **onclick()** - [Function], Called when the item is clicked.

"control":

Enables to configure the mouse and touch control properties and allows to capture events sent by the panorama object.

Properties:

- **mousemode** - [String], Type of control for mouse devices. Possible values: 'move', the camera pans in the direction the mouse is moved. 'drag' the image/panorama can be dragged with the mouse.
- **touchmode** - [String], Type of control for touch devices. Possible values: 'move', the camera pans in the direction the finger is moved. 'drag' the image/panorama can be dragged with the finger.
- **zoomtocursor** - [Boolean], Use the position of the mouse cursor as center when zooming in and out (default: true)
- **mwheel** - [Boolean], Allow to use the mouse-wheel for zooming (default: true)
- Events on the global panorama layer will be sent to the control object. The events are inherited from layer: chap. 5.2.3 (→ p.31)
- **propagatelaast** - [Boolean], By default mouse events are sent to the global panorama layer first, then to all objects in front beginning at the topmost. Set this attribute to true to sent the events first to the objects in front (default: false)
- **enablevrcontroller** - [Boolean], Uses a controller or gamepad, if available and VR mode is active (default: true)

- `vrdisplay` - [String], Type of the VR display. Possible values: 'auto', use WebXR or the older WebVR if supported by the browser, otherwise an emulated split screen display is used, for example on smartphones used with Google Cardboard. 'splitscreen', always use an emulated split screen display, no WebXR or WebVR. (default: auto)

"data":

In the `data` object you can dynamically store arbitrary key-value pairs. This allows to store data at one location and access that data from multiple other objects.

Properties:

- Arbitrary key value pairs in JSON format.
- Properties inherited from object: chap. 4.3.1 (→ p.27)

Example:

```
"data": {  
  "id": "globalData",  
  "messageText": "Hello, World!"  
}  
  
//Access the data from within a function:  
console.log(this.viewer.get('globalData').messageText);
```

"events":

In the `events` object you can assign functions to events. This way you can execute own actions on certain events.

Properties:

- [Events] - You can assign here a function to each event defined in chap. 5.2.2 (→ p.29).
- Properties inherited from object: chap. 4.3.1 (→ p.27)

Example:

```
"events": {  
  "id": "mainEvents",  
  "onplay": function(){ console.log('auto play started'); }  
}
```

"gallery":

The `gallery` object is an array with items for a gallery of thumbnail images which are linked to other panoramas or other views in the same panorama.

A gallery is displayed with a `galleryElement` object.

A gallery object can also be used to show the locations of panoramas on a map with the `mapElement`.

Properties and functions:

- `/*int*/ current()` - [Function] Returns the index of the selected gallery item.
- `/*int*/ previous()` - [Function] Returns the index of the gallery item to the left, returns the rightmost index, if the current index is 0.
- `/*int*/ next()` - [Function] Returns the index of the gallery item to the right, returns 0, if the current index is the rightmost.
- `set(/*int*/ i)` - [Function] Selects the item with index *i*.
- `viewer` - [Object], A variable to access the Viewer instance (see chap. 5.1 (→ p.29))
- `/*Object*/ get(/*String*/ id)` - [Function] Find a layer or object based on its id.

Properties of each array item:

- `href` - [String] Url of the parameter file of the destination panorama (.js or .json).
- `thumbnail` - [String] Url of a JPG preview image.
- `view` - [Object] An optional view object to define initial view of the destination. See chap. 4.2 (→ p.26).
- `location` - [Object] An optional location object to define the location of the destination. See chap. 4.2 (→ p.21).
- `text` - [String] HTML formatted text displayed as description for the gallery item.
- `tooltip` - [Object] A textbox object as tooltip for the gallery item (see `textbox` object). Here the position of the tooltip is always attached to the mouse when the mouse hovers over the item and cannot be declared with the corresponding `textbox` property.

"galleryElement":

A `galleryElement` is a thumbnail overview which can provide links to different views in the same panorama or links to other panoramas.

It can be a stand-alone element in the Viewer or part of a `buttonBox` element. A `galleryElement` displays the content of the gallery object.

Properties:

- `thumbnailWidth` - [int] Width of the thumbnails (default: 96).
- `thumbnailHeight` - [int] Height of the thumbnails (default: 96).
- `thumbnailPadding` - [int] Set the padding of the thumbnails in pixels (default: 8).
- `framewidth` - [int] Width of the frame around a selected thumbnail (default: 3).
- `selectioncolor` - [String] Color of the frame around a selected item (default: #fff).
- `selectionstyle` - [String] Equals the CSS *style* attribute in HTML. Set the style of the frame around a selected thumbnail.
- `collapsed` - [Boolean] Set the status of the gallery element. If `true`, then the gallery is hidden.
- `style` - [String] Set the style of the gallery element, for example the background color or border.
- `thumbstyle` - [String] Set the style of the gallery thumbnails.
- `thumbstylehover` - [String] Same as `thumbstyle`, applied when the mouse hovers over the thumbnail.

- **thumbstyleactive** - [String] Same as **thumbstyle**, applied when the mouse is pressed over the thumbnail.
- **contentstyle** - [String] Set the style of the content of the gallery thumbnails - the content is the overlay defined in the **text** attribute of each gallery item.
- **contentstylehover** - [String] Same as **contentstyle**, applied when the mouse hovers over the thumbnail.
- **contentstyleactive** - [String] Same as **contentstyle**, applied when the mouse is pressed over the thumbnail.
- Properties inherited from layer: chap. 4.3.2 (→ p.27)
- Events inherited from layer: chap. 5.2.3 (→ p.31)

"gfx":

Allows loading of bitmap images, which are then available for the composition of buttons.

Properties:

- **src** - [String], Image Url.
- Properties inherited from object: chap. 4.3.1 (→ p.27)

"hotspot":

Describes a hotspot in the panorama or zoom image

Properties:

- **anchor** - [Number,Number], For floating hotspots, the pixel coordinates in the hotspot image at which the hotspot should be anchored in the panorama. [0,0] is the top left corner of the image. By default, the anchor is the center of the hotspot image.
- **bitmap** - [String], URL of an image file used as bitmap hotspot (JPG, GIF, PNG, WEBP).
- **bitmapttype** - [String], Display type of the bitmap hotspot. Possible values: **floating**, the image will be displayed floating over its position in the panorama or **embedded**, the image will be displayed spatially embedded into the panorama (default: **floating**).
- **bitmapscale** - [Number;Number(optional)], Size of a floating bitmap hotspot. The first value is the scale in the default state, the optional second value is the scale when the mouse is hovering over the hotspot, i.e. '1.0;1.2' to enlarge the hotspot by 20 percent when the mouse hovers over the hotspot.
- **displaytext** - [String], Display mode for the tooltip. 'permanent' shows the tooltip permanently, 'onhoverstatic' shows the tooltip on a fixed position when the mouse hovers over the tooltip, 'onhoverfollow' shows the tooltip when the mouse hovers over the hotspot attached to the mouse cursor.
- **embeddedwidth** - [Number], Width of an embedded bitmap hotspot (see unit)
- **href** - [String], Destination URL. URLs are links to webpages or other panoramas. To link to another panorama enter the URL of the parameter file (.js or .json) of the panorama (see also target)

- **opacity** - [Number;Number(optional)], Opacity of a hotspot. The first value is the opacity in the default state, the optional second value is the opacity when the mouse is hovering over the hotspot. Valid values are in the range from 0 (invisible) to 1 (opaque). For example '0.5;1.0', to switch the hotspot from semitransparent to opaque when the mouse hovers over the hotspot.
- **position** - [Number,Number], Position of a bitmap hotspot (see **unit**).
- **rotate** - [Number], Rotation of an embedded bitmap hotspot in degrees.
- **target** - [String], HTML target. If using the keyword `myself` here, then the `href` attribute expects a parameter file (.js or .json) of another panorama which will be opened in the same Viewer instance, else this attribute has the same meaning as in HTML.
- **targetview** - [String], Command list, which allows to set an initial view for the new panorama (see `cmd` in `openPanorama` in chap. 5.3.2 (→ p.37))
- **transition** - [String], Command list, where you can configure a transition effect when opening another panorama. Specification based on an example: 'zoomin,1.0 ; blend,2.0' Here the `zoomin,1.0` initiates a zoom on the clicked hotspot within a duration of 1.0 seconds, following a fade into the new panorama (`blend`) within 2.0 seconds.
- **unit** - [String], Unit of the coordinates of the **position** property. Possible values: `deg`, for coordinates in degrees, `norm` for normalized values in the range from 0 to 1. For panoramas only `deg` is allowed, for planar zoom images only `norm`.
- Properties inherited from `layer`: chap. 4.3.2 (→ p.27)
- Events inherited from `layer`: chap. 5.2.3 (→ p.31)

Nested objects:

- **textbox** - Shows a `textbox` element as tooltip for the hotspot (see also "textbox" object). If a `textbox` is used as tooltip, then the position of the tooltip is not declared with the corresponding `textbox` properties, but with the `displaytext` property of the hotspot.
- **animation** - Specifies an animated hotspot. The frames of the animation are stored as `stripe` or `matrix` in the `bitmap` property of the hotspot. The frames must be ordered in the `bitmap` image in lines from left to right and top to bottom.
 - **width** - [int], Width of each frame in pixels.
 - **height** - [int], Height of each frame in pixels.
 - **frames** - [int], Total number of frames.
 - **delay** - [Number], Delay in seconds until the animation starts after the scene is loaded.
 - **iterations** - [int], Specifies how many times the animation is played. Set to 0 for infinite iterations.
 - **frameOffset** - [int], First frame of the animation. The animation stops also at this frame after a `mouseover` or after playing a limited number of iterations.
 - **duration** - [Number], Time for one animation loop in seconds.
 - **mouseover** - [Boolean], The animation plays only when the mouse hovers over the hotspot.

"hotspots":

Properties of hotspots.

Properties:

- **visible** - [Boolean], If set to true the hotspots are always visible, else only when the mouse cursor hovers over a hotspot (default: true)

"image":

Specification of the image data. The `image` object and the image data should be created automatically using the PanoramaStudio application. You should not edit or change this section.

Properties:

- **projection** - [String], Type of the image projection.
Possible values: 'cubic', 'planar'
'planar' Shows the panorama or image as flat zoom image. 'spherical', 'cubic' shows the image data as 3D panorama.
- **multilevel** - [Boolean], true, if the "image" object contains multiple resolutions, else false.
- **baseindex** - [Number], Base index of the image tile numbering
- **resolutionadjust** - [Number], configures when the view switches from one resolution to the next. Allowed values are 0.5 to 1.0, where 0.5 means switching early to the next higher resolution which results in a sharper image but requires more resources and bandwidth for the image data. (default: 0.82, min: 0.5, max: 1.0)
- **mipmapping** - [Boolean], Use mipmapping to improve texture quality in WebGL. (default: false)

Nested objects:

- **bitmap** - image data of the panorama or zoom image. If "multilevel": true then the Viewer expects a list of bitmap objects of the image in different resolutions.

Properties:

- **width** - [Number], Width of the overall image for this bitmap object
- **height** - [Number], Height of the overall image for this bitmap object
- **tilesize** - [Number], Width/height of an image tile in this bitmap object, if the image is split up in tiles.
- **src** - [String]. URL of a JPG image file or a template for a series of JPG image tiles. If the image is split up in tiles, then %x and %y is used as placeholder for the x and y coordinates in the URL of the image tile. Leading zeros i.e. for a three digit number are expressed with %00x.

Example:

```
"bitmap" : {
  "width": 22705,
  "height": 7918,
  "tilesize": 841,
  "src": "pano1/pano1.tiles/14\_\\%0y\\\_\\%0x.jpg"
}
```

Nested objects of bitmap:

- "left", "right", "front", "back", "up", "down" - In a cubic panorama the bitmap object expects the six cube faces in this nested objects. The src property contains the image file name or the template for the image tiles as otherwise in the src property of the bitmap object. width and height in the parent bitmap object must be identical and describe the size of a cube face.
- preview - Preview image

Properties:

- src - [String], Path of a JPEG image file
- rotate - [String], For cubic panoramas the preview image must be a horizontal strip of the six cube faces. The rotate attribute contains optionally an encoded instruction on how to rotate the six cube faces. It must be a string of 6 characters. 0 means no rotation for that face, 1: 90 degrees, 2: 180 degrees, 3: 270 degrees.
Example:

```
"rotate": "001122"
```

"include":

include allows to include additional parameter files. An include file has the same structure as the main parameter file. Within the "panoStudioViewer" object you can declare properties and objects to configure the currently displayed panorama. This allows to store settings and parameters used by several panoramas in one common include file.

Properties:

- src - [String], URL of the file to include.

Example:

```
"include": {
  "src": "fileToInclude.js"
}
```

"key":

Element containing the license key of a Single-Domain-License. On the other hand, the license key of a Multi-Domain-License is stored within the Viewer file by PanoramaStudio.

Example:

```
"key": "649303968742"
```

"lensflare":

An artificial *lens flare* in a 3D panorama

Properties:

- **pan** - [Number], Horizontal position of the lens flare in degrees (min: 0, max: 360)
- **tilt** - [Number], Vertical position of the lens flare in degrees (min: -90, max: 90)
- **type** - [Number], Type of the lens flare. There are 7 pre-defined types (min: 0, max: 6)
- **brightening** - [Number], Brightening of the scene if the camera points to this light source (default: 0.8, min: 0, max: 1)
- **scale** - [Number], Size of the glares for this lens flare (default: 1, min: 0.1, max: 10)
- Properties inherited from object: chap. 4.3.1 (→ p.27)

"location":

A *location* object contains optional location data of the scene. The *location* object can be part of a *node* object or a *gallery* entry.

Properties:

- **heading** - [Number] Heading of the center of the panoramic image in degrees. 0 is north. The heading is used for the compass/radar feature and for the radar option on the map. (default: 0).
- **lat** - [Number] Latitude as decimal number (default: 0).
- **lon** - [Number] Longitude as decimal number (default: 0).

"logo":

The *logo* object allows to insert an own logo replacing the *PanoramaStudio Viewer* logo, if the Viewer runs on a local computer or online with a license for commercial use (see key)

Properties:

- **src** - [String], URL of a logo image file (GIF, PNG, or JPEG).
- **href** - [String], Target URL of the webpage opened when the logo is clicked.
- **target** - [String], HTML target.
- Properties inherited from *layer*: chap. 4.3.2 (→ p.27)
- Events inherited from *layer*: chap. 5.2.3 (→ p.31)

Nested objects:

- **textbox** - Shows a `textbox` object as tooltip for the logo (see `textbox` object). Here the position of the tooltip is always attached to the mouse when the mouse hovers over the logo and cannot be declared with the corresponding `textbox` property.

"mapElement":

A `mapElement` contains a map to show the locations of a virtual tour. The entries of a `gallery` object will be visualized on the `mapElement` with markers, if the these entries contain the required location data. A map can be a stand-alone element in the `Viewer` or part of a `buttonBox` element.

The `Viewer` uses *Google Maps* and embeds the provided map with with the *Google Maps JavaScript API*.

To use the Google Maps platform you need an *API key* which you can request at Google. Depending on the number of page views, costs for the use of the *Maps JavaScript API* may be incurred at Google. However, most use cases will be free.

Under the following link you find further information and the option to request an API key at Google:
<https://developers.google.com/maps/documentation/javascript/get-api-key?hl=en>

Properties:

- **gmapstype** - [String], Map types.
Possible values are "roadmap" for roadmap view, "satellite" for satellite view, "hybrid" for satellite view with street names, "osm" to use OpenStreetMap data with the Google Maps API.
- **gmarker** - [String], Replace the default marker with a custom marker. `gmarker` is an icon object as specified in the Google Maps API. See https://developers.google.com/maps/documentation/javascript/markers?hl=en#complex_icons

Example:

```
"{ url: 'myMarker.png', anchor: new google.maps.Point(15, 42) }"
```

- **gmarkeractive** - [String], Replace the default active marker with a custom marker.
See `gmarker`.
- **apikey** - [String], A *Google Maps JavaScript API key* which allows to embed Google Maps in your website or in interactive panoramas on your website (required).
See <https://developers.google.com/maps/documentation/javascript/get-api-key?hl=en>
- **zoom** - [int], Initial zoom level of the map. The zoom can be set from 0 (world) to 18 (maximum zoom). Some regions allow a zoom of up to 21. (default: 18, min: 0, max: 21)
- **zoomcontrols** - [Boolean], Display controls for changing the zoom level. (default: true)
- **typecontrols** - [Boolean], Display controls to choose a map type. (default: true)
- **onmarkerclick** - [Function], Called when a marker was clicked. Allows to execute further actions besides the automatically performed call of the the destination panorama of the marker.
- **width** - [String], Width of the map in percent (%) or pixels (px) if the map is not part of a `buttonBox`. Example:

```
"width": "50%"
```

```
"width": "500px"
```

- **height** - [String], Height of the map in percent (%) or pixels (px).
- **style** - [String] Set the style of the map element, for example the background color or border.
- **contentstyle** - [String] Set the style (e.g. position and size) of the map within the map element - the content is the Google maps overlay used for the map.
- Properties inherited from layer: chap. 4.3.2 (→ p.27)
- Events inherited from layer: chap. 5.2.3 (→ p.31)

Nested objects:

- **radar** - Shows an orientation element in radar style on the active marker. See orientation (chap. 4.2 (→ p.23)).

"node":

A node is a container for a scene or panorama, respectively. If the individual panoramas of a virtual tour are stored in a node, then controls and other objects can be used for the whole virtual tour.

The following objects can be stored in a node:

- image
- camera
- view
- autoplay
- video
- audio
- hotspots
- hotspot
- lensflare
- data
- location
- logo

"orientation":

The orientation object shows the viewing direction in form of a radar or compass element.

Properties:

- **type** - [String], compass to show a compass element, radar to show a radar element (default: compass).
- **radius** - [Number], Size of the radar within the element (default: 0.9, min: 0, max: 1).
- **rotatedial** - [Boolean], true, to rotate the dial in the compass view (the needle is fixed), false to rotate the needle of the compass (default: false).
- **skindial** - [String], Describes how the compass dial or the radar background is painted. See chap. 4.3.3 (→ p.28).
- **skinneedle** - [String], Describes how the compass needle is painted. See chap. 4.3.3 (→ p.28).
- **skinframe** - [String], Describes how the fixed frame of the compass or radar element is painted. See chap. 4.3.3 (→ p.28).
- **radarcontext** - [Function], Function called with the 2D canvas context of the radar. Allows to modify attributes such as `lineWidth`, `fillStyle`, or `strokeStyle` of the context before the radars is painted.

"settings":

The settings object allows to configure the behavior when loading image data.

Properties:

- **loadqueues** - [Number], Maximum number of simultaneous image tile downloads (default: 4, min: 1, max: 10)
- **crossOrigin** - [String], Optionally set the crossOrigin attribute of the `<img>` elements used by the Viewer in the DOM (default: undefined, when running locally, else Anonymous)
- **backgroundcolor** - [#RRGGBB or #RGB], Background color as hexadecimal color code. Used in zoom images if the image does not fill the entire Viewer and the background becomes visible (default: #000000).
- **safeareamargin** - [String], Correction value in pixels which is additionally added or subtracted to the position of elements at the image edges in case of applying an extension of the display area (see `viewport_fit_cover`, chap. 5.3.1 (→ p.33)). Order and application identical to padding in CSS (top, right, bottom, left). Negative values can be used to prevent elements from being indented too far. Example:

```
"safeareamargin": "-8 -8 -8 -8"
```

"textbox":

The `textbox` object displays text and other HTML content. It is used amongst others as tooltip for the hotspot and logo objects, but can also be used directly for displaying text in the panorama.

Properties:

- **text** - Can contain arbitrary HTML for text and other elements.
- **alpha** - [Number], transparency of the textbox, 0: completely transparent, 1: opaque (default: 1.0, min: 0.0, max:1.0)
- **background** - [Boolean], show text in front of a background (default: false)
- **backgroundcolor** - [Number], background color in hexadecimal or HTML notation.
- **width** - [String], fixed width for the textbox, even if the text is smaller than width. Valid entries in percent (%) or pixels (px). Example:

```
"width": "33%"
```

```
"width": "250px"
```

- **maxwidth** - [String], if width is not defined, you can define with maxwidth a maximum width. Especially for a single-line text with background which is smaller than maxwidth the width of the textbox will be adjusted. For longer texts maxwidth causes an automatic line break. Valid entries in percent (%) or pixels (px).
- Properties inherited from layer: chap. 4.3.2 (→ p.27)
- Events inherited from layer: chap. 5.2.3 (→ p.31)

"toolbar": (deprecated)

The out-dated `toolbar` element is replaced by `buttonBox` and `button`. If the current version of the Viewer finds a `toolbar` element in an old parameter file (PanoramaStudio Viewer version < 4), then it uses instead a default set of buttons based on `buttonBox` and `button`.

"translate":

The `translate` object allows to translate text according to the language setting of the browser. Therefore it can contain objects named after language codes. For example `en-US`, `de` or `de-AT`.

These language code objects can contain key value pairs, where in each pair the key is the text to translate and the value is the translated text.

The Viewer provides for the translation the function `tr()`, which uses the `translate` object to translate text according to the browser language. See `tr()`, p. 39.

Example:

```
"translate": {  
  "de": {  
    "Cancel": "Abbrechen"  
  }  
}
```

"video":

Embed a video file.

Properties:

- **src** - [String], URL of a MP4 video file
- **src_alt** - [String], Optional URL of a video file encoded in the WebM format, as alternative video source for browsers which do not support MP4.
- **poster** - [String], Specifies an image to be shown until the playback is started.
- **loop** - [Boolean], If true, the video will start over again, every time it is finished (default: true).
- **autoplay** - [Boolean], Plays the video file automatically as soon as it is loaded when set to true (default: false).
- **videowidth** - [Number], Width of the source video in pixels.
- **videoheight** - [Number], Height of the source video in pixels.
- **width** - [Number], Width of the video element in the panorama (see unit).
- **position** - [String], Position of the video element (see unit).
- **rotate** - [Number], Rotation of the video in degrees.
- **unit** - [String], Unit of the coordinates in the position property and the size in the width property. Possible values: deg, for coordinates in degrees, norm for normalized values in the range from 0 to 1. For panoramas only deg is allowed, for planar zoom images only norm.
- **volume** - [Number], Audio volume of the video (default: 1.0, min: 0.0, max: 1.0).
- Properties inherited from object: chap. 4.3.1 (→ p.27)
- Events inherited from layer: chap. 5.2.3 (→ p.31)

Example:

```
"video": {  
  "src": "video.mp4",  
  "loop": true,  
  "position": "160.0,-6.0",  
  "rotate": -23.0,  
  "unit": "deg",  
  "width": 33.0,  
  "autoplay": false,  
  "volume": 1.0,  
  "videowidth": 1280,  
  "videoheight": 720  
}
```

"view":

Sets the initial view for the panorama or zoom image

Properties:

- **pan** - [Number], Horizontal viewing direction in the panorama in degrees (default: 0)
- **tilt** - [Number], Vertical viewing direction in the panorama in degrees (default: 0)
- **hfov** - [Number], Horizontal field-of-view in the panorama in degrees (default: 90)
- **x** - [Number], Horizontal position of the viewer in a zoom image in the range of 0 (left) to 1 (right) (default: 0.5)
- **y** - [Number], Vertical position of the viewer in a zoom image in the range of 0 (top) to 1 (bottom) (default: 0.5)
- **zoom** - [Number], Zoom setting for zoom images. A value of 0 shows the image completely zoomed out, a value of 1 shows the image in its original size (default: 0)
- **mode** - [String], Mode for zoom images, `fillWindow`: the image can be zoomed out as far as it still fills the window or `fitInWindow`: The image can be zoomed out until it fits completely in the window (default: `fitInWindow`)
- **shortdescription** - [String], short description used for the marker on the map and the gallery (same as text in a gallery entry).

4.3 All-purpose properties

Objects inherited from `object` and `layer` have some properties in common:

4.3.1 Properties of all objects

- **id** - [String], assign an unique ID to an object. For example to access the object with the `get()` function.

4.3.2 Properties of all layers

Properties of layers and all objects inherited from `layer`:

- **viewer** - [Object], A variable to access the Viewer instance (see chap. 5.1 (→ p.29))
- **visible** - [Boolean], Visibility of the layer (default: `true`)
- **checkHover** - [Boolean], Layer captures mouse movements and sends corresponding events.
- **captureMouse** - [Boolean], If set to `true` then all mouse events will be sent to this layer when the mouse is pressed on the layer until the mouse is released, even when the mouse is moved outside of the layer.
- **style** - [String], Equals the CSS *style* attribute in HTML. The object can be styled this way with CSS.
- **align** - [String], Alignment of an element. Possible values: `left`, `top`, `right`, `bottom`, `lefttop`, `leftbottom`, `righttop`, `rightbottom` und `center`
- **xoff** - [Number], Offset of the object to the left (<0) or to the right (>0) in pixels.
- **yoff** - [Number], Offset of the object to the top (<0) or to the bottom (>0) in pixels.
- **padding** - [Number], Padding as used in CSS in pixels (default: 0)
- Properties inherited from `object`: chap. 4.3.1 (→ p.27)

4.3.3 Properties of Button and Orientation

- **skin** - [String], List of instructions, which allows to copy images on the layer and modify these images with some filters. The following instructions are supported:
 - `dim(String dstx,String dsty,String dstw,String dsth,int srcw,int srch)`
Set a size for the canvas on the layer, if this size is different from the layer size.
`dstx, dsty`: X/Y coordinate of the top left corner of the canvas on the layer in pixels (px) or percent (%).
`dstw, dsth`: Width/height of the canvas on the layer in pixels (px) or percent (%).
`srcw, srch`: Width/height of the internally used canvas.
 - `copy(int gfxId,int srcx,int srcy,int srcw,int srch,int dstx,int dsty,int dstw,int dsth,String tintcolor[optional],String actionId[optional])`
Copies a segment from a gfx object onto the layer's canvas.
`gfxId`: Id of the gfx object used as source. The Viewer contains at defaultSkin already an integrated bitmap graphic for some default buttons.
`srcx, srcy`: X/Y coordinate of the top left corner of the rectangle in the graphic to copy.
`srcw, srch`: Width/height of the rectangle in the source graphic.
`dstx, dsty`: X/Y coordinate of the top left corner of the button's destination canvas.
`srcw, srch`: Width/height of the layer's destination rectangle.
`tintColor`: Optionally, you can tint the copied graphic with a color. `tintColor` specified in CSS format, e.g. blue, #ff0000 or rgb(255,0,0)
`actionId`: Optionally, you can directly modify the canvas context. `actionId` specifies the Id of an action which is called when the canvas is initialized. The first parameter of the called action is the canvas context.
 - `comp(Number alpha,String composite[optional])`: Sets alpha or transparency value and compositing operation of the following copy instructions.
`alpha`: Value between 0.0 (completely transparent) and 1.0 (opaque) (default: 1.0).
`composite`: Compositing operation to apply when using copy. Equals `globalCompositeOperation` of the HTML *canvas* element (default: source-over). For possible values see for example:
http://www.w3schools.com/tags/canvas_globalcompositeoperation.asp.
 - `shadow(Number blur,Number xoff,Number yoff,String color)` Apply a shadow filter on following copy instructions.
`blur`: Unsharpness in pixels.
`xoff`: Horizontal offset of the shadow in pixels.
`yoff`: Vertical offset of the shadow in pixels.
`color`: Shadow color. Color specified in CSS format, e.g. blue, #ff0000 or rgb(255,0,0)

Example:

```
"skin": "dim(-4px,-4px,32px,32px,32,32);comp(0.6);
copy(defaultSkin,0,128,64,64,0,0,30,30);shadow(3,0,0,rgba(0,0,0,1));
comp(1);copy(defaultSkin,128,192,64,64,4,4,24,24)"
```

- Properties inherited from layer: chap. 4.3.2 (→ p.27)

5 JavaScript API

5.1 Access a panorama object

If a DIV element contains a *PanoramaStudio Viewer* object, then you can access it with the variable `panoStudioViewer`. Accordingly, you get the Viewer object e.g. with:

```
var viewer = document.getElementById("panoStudioViewerID").panoStudioViewer;
```

5.2 Events

5.2.1 Overview

The PanoramaStudio Viewer provides a number of events, which can be used to execute actions when a certain event happens.

Events can be captured in several ways.

- Externally in the webpage:

```
var viewer = document.getElementById("panoStudioViewerID").panoStudioViewer;
viewer.oninit = function(){ /* insert your code here */ }
```

- Global events of a Viewer instance in the `events` object of the parameter file. Functions are specified as strings in the parameters files of the Viewer:

```
"events": {
  "id": "mainEvents",
  "oninit": function(){ /* insert your code here */ },
},
[...]
```

- Local events of a layer in the corresponding object. As example an onclick event on a button:

```
"button": {
  "onclick": function(){ /* insert your code here */ },
  [...],
},
[...]
```

5.2.2 Events of a `panoStudioViewer` instance

A `panoStudioViewer` object provides the following events:

```
onenterfullscreen()
```

Called when the panorama has been switched to the fullscreen mode.

```
onentervrmode()
```

Called when the Viewer enters the VR mode.

```
onexit()
```

Called before a panorama is left.

```
onexitfullscreen()
```

Called when the panorama left the fullscreen mode.

```
onexitvrmode()
```

Called when the Viewer exits the VR mode.

```
onframe(frameTime/*Number*/,viewChanged/*Boolean*/)
```

Called for each frame.

Optional parameters:

frameTime, Time in milliseconds since last frame.

viewChanged, true if the view in the panorama has changed.

Example:

```
viewer.onframe = function(frameTime,viewChanged){  
  console.log("frame time (ms): "+frameTime+" view changed: "+changed); }
```

```
ongyroscopetoggle(enabled/*Boolean*/)
```

Called when the status of the gyroscope control changes. enabled signals if the gyroscope control is now enabled or disabled.

```
oninactive(ms/*Number*/)
```

Called after 2 seconds of user inactivity. The call is repeated every 2 seconds of user inactivity.
ms, time in milliseconds since last user interaction.

```
oninit()
```

Called when a panorama was initialized.

```
onplay()
```

Called when the auto play starts.

```
onresize()
```

Called when the size of the Viewer element has changed.

```
onscenechanged()
```

Called when the view has switched to another panorama.

```
onstartaudio(id/*String*/)
```

Called when the playback of an audio object starts.
id, ID of the audio object.

```
onstop()
```

Called when the auto play stops.

```
onstopaudio()
```

Called when the playback of an audio object stops.
id, ID of the audio object.

```
onusewebgl(available/*Boolean*/)
```

Called when the Viewer is initialized. Reports if WebGL is used. Optional parameters:
available, is *true*, if WebGL is used for displaying the content, otherwise CSS3D is used.

```
onviewchanged(mouseX/*Number*/,mouseY/*Number*/)}
```

Called when the view in the panorama was changed.
Optional parameters:
mouseX, mouseY, X/Y coordinates of the cursor.

```
onvisibilitychange()
```

Is called when the viewer or browser tab is shown or hidden.

5.2.3 Local events on each layer

```
/*Boolean*/ onclick(/*Number*/ mouseX,/*Number*/ mouseY)
```

Called on a mouse click or touch tap.
mouseX, mouseY, X/Y coordinates of the mouse relative to the Viewer window. Return *true*, if the events has been processed, then it will not be sent to other objects.

```
/*Boolean*/ ondblclick(/*Number*/ mouseX,/*Number*/ mouseY)
```

Called when a double click/tap took place.
mouseX, mouseY, X/Y coordinates of the mouse relative to the Viewer window. Return *true*, if the events has been processed, then it will not be sent to other objects.

```
/*Boolean*/ ondown(/*Number*/ mouseX,/*Number*/ mouseY)
```

Called when the left mouse button is pressed or a touch event started.
mouseX, mouseY, X/Y coordinates of the mouse relative to the Viewer window. Return *true*, if the events has been processed, then it will not be sent to other objects.

```
onexit()
```

Called before an object is deleted.

```
onframe(frameTime/*Number*/,viewChanged/*Boolean*/)
```

Called for each frame.

Optional parameters:

frameTime, Time in milliseconds since last frame.

viewChanged, true if the view in the panorama has changed.

```
oninit()
```

Called when an object was initialized.

```
/*Boolean*/ onmove(/*Number*/ mouseX,/*Number*/ mouseY)
```

Called when the mouse is moved over an object.

mouseX, mouseY, X/Y coordinates of the mouse relative to the Viewer window. Return true, if the events has been processed, then it will not be sent to other objects.

```
onout(/*Number*/ mouseX,/*Number*/ mouseY)
```

Called when the mouse leaves the region covered by the object.

mouseX, mouseY, X/Y coordinates of the mouse relative to the Viewer window.

```
onover(/*Number*/ mouseX,/*Number*/ mouseY)
```

Called when the mouse enters the region covered by the object.

mouseX, mouseY, X/Y coordinates of the mouse relative to the Viewer window.

```
onresize()
```

Called when the size of the Viewer element has changed.

```
onup(/*Number*/ mouseX,/*Number*/ mouseY)
```

Called when the left mouse button is released or a touch event ended and if the object captures the mouse.

mouseX, mouseY, X/Y coordinates of the mouse relative to the Viewer window.

```
onviewchanged(mouseX/*Number*/,mouseY/*Number/>)}
```

Called when the view in the panorama was changed.

Optional parameters:

mouseX, mouseY, X/Y coordinates of the mouse.

5.3 Functions

5.3.1 Functions of the global, static panoStudioViewer object

If the PanoramaStudio Viewer based on the `panoStudioViewer.js` script is loaded in a webpage, then there is a global, static variable `panoStudioViewer` available to insert and remove panoramas.

```
insert(/*String*/ id,/*String*/ parameterFile,/*Object*/ inlineParameters,  
      /*String*/ startupCallback)
```

Adds a panorama defined by the parameter file `parameterFile` to the DIV element with the unique ID `id`.

`inlineParameters` is an optional list of parameters to configure the Viewer at startup. Possible values are:

- `{html5:disable_webgl_warning}`, do not show a warning, if the browser does not support WebGL.
- `{html5:nowebgl}`, use CSS-3D transforms instead of WebGL.
- `{html5:viewport_fit_cover}`, if the Viewer is configured to fill the browser window or displayed in fullscreen mode, then this option expands the panorama further to fill also the edges on devices with rounded display corners or a notch in the display which is otherwise not covered.

Or any combination of the above:

- `{html5:nowebgl|disable_webgl_warning}`

`startupCallback` is an optional function called after the initialization of the Viewer. The argument must be a string containing a JavaScript function in the following form:

```
"function(){ /* insert your code here */ } "
```

```
remove(/*String*/ id)
```

Remove a panorama based on its unique `id` from the containing DIV element.

5.3.2 Functions of the Viewer instance

If a DIV element contains a panorama, then you can access this viewer instance with the local variable `panoStudioViewer` (not to be confused with the global, static `panoStudioViewer` object).

```
var viewer = document.getElementById("panoStudioViewerID").panoStudioViewer;
```

The following functions are available:

- `action(...)`
- `add(...)`
- `camera()`
- `checkForGyroscope()`
- `currentRuntime()`
- `enableGyroscope(...)`
- `enterVRMode()`

- `exitVRMode()`
- `fullEquirectangular()`
- `fullscreenSupported()`
- `gallery()`
- `get(...)`
- `getView()`
- `gyroscopeEnabled()`
- `height()`
- `hideCursor()`
- `hotspots()`
- `inactivePeriod()`
- `isMobile()`
- `isPlaying()`
- `isVRModeEnabled()`
- `lensflares()`
- `location()`
- `lock()`
- `map()`
- `maxZoom()`
- `openPanorama(...)`
- `panoType()`
- `panTo(...)`
- `remove(...)`
- `safeArea()`
- `screenToPano()`
- `setView(...)`
- `showCursor()`
- `startAutoPlay()`
- `stopAutoPlay()`
- `systemInfo()`
- `tr(...)`
- `toggleFullscreen()`
- `unlock()`
- `versionInfo()`
- `width()`
- `zoom()`

```
/*var*/ action(/*String*/ id, arg1, arg2, ...)
```

Searches in the actions objects (chap. 4.2 (→ p.10)) for a function with corresponding id and calls the function with the arguments (arg1, arg2, ...). If the called function returns a value, then action() returns this value.

```
add(/*String*/ type,/*Object*/ json)
```

Adds a layer of type button, buttonBox, audio, video or textbox.
json contains the JSON object with the properties required for the layer type to add.

Example, show an *info box* on a click on a button:

```
"button" : {
  [...]
  /* Add a textbox; The JSON object 'textElement' of the textbox is also included here */
  "onclick": function(){
    var o = this.get('infobox');
    if (!!o){ this.viewer.remove('infobox'); }

    else {

      this.viewer.add('textbox',this.textElement);

    }
  },
  "textElement": {
    [...]
    "id": "infobox",
    "onclick": function(){ this.viewer.remove('infobox'); } ,
    "text": "My Textbox",
    [...]
  },
  [...]
}
```

```
/*Object*/ camera()
```

Returns an object with the parameters of the camera object (minpan, maxpan, ...).

```
/*Boolean*/ checkForGyroscope()
```

Returns true, if a gyroscope (motion sensors) is available to control the panorama.

```
/*Number*/ currentRuntime()
```

Returns the runtime of the current scene in milliseconds.

```
/*Boolean*/ enableGyroscope(/*Boolean*/ on)
```

Call with true or false to enable or disable the gyroscope control.
Returns the new status.

```
/*Boolean*/ enterVRMode()
```

Enters the VR mode. Returns, if the VR mode could be enabled.

```
exitVRMode()
```

Leaves the VR mode.

```
/*Boolean*/ fullEquirectangular()
```

Returns *true*, if the current panorama is a 360x180 panorama, else returns *false*.

```
/*Boolean*/ fullscreenSupported()
```

Returns *true*, if the fullscreen mode is supported by the browser.

```
/*Array*/ gallery()
```

Returns the gallery array, if defined. See chap. 4.2 (→ p.15).

```
/*Object*/ get(/*String*/ id)
```

Find an object based on its id.

```
/*Object*/ getView()
```

Returns an object with the angles of the current viewing direction and field-of-view (pan, tilt, hfov). In a planar zoom image the object contains the current center of view and zoom (x, y, zoom).

```
/*Boolean*/ gyroscopeEnabled()
```

Retrieves the status of the gyroscope. Returns *true*, if the gyroscope control is enabled.

```
/*Number*/ height()
```

Current height of the Viewer in pixels.

```
hideCursor()
```

Hides the mouse cursor.

```
/*Array*/ hotspots()
```

Returns an array of the hotspots in the panorama.
The hotspots array object provides some additional functions:

- */*Object*/* get(*/*int*/* i), Returns a JSON object of the hotspot at index position i.
- *remove*(*/*int*/* i), Removes the hotspot at index position i.

- `insert(/int/ i, /Object/ h)`, Inserts a hotspot `h` as JSON object (see also chap. 4.2 (→ p.18)) at index position `i`.
- `enableDrag(/Boolean/ b)`, Activates or deactivates the option to move hotspots with drag&drop.

```
/Number/ inactivePeriod()
```

Returns the duration since the last user interaction in milliseconds.

```
/Boolean/ isMobile()
```

Returns true if the Viewer runs on an Android or iOS mobile device.

```
/Boolean/ isPlaying()
```

Returns true, if the auto play is running.

```
/Boolean/ isVRModeEnabled()
```

Returns, if the Viewer is in VR mode.

```
/Array/ lensflares()
```

Returns an array of the lensflares in the panorama.

The lensflares array object provides some additional functions:

- `/Object/ get(/int/ i)`, Returns a JSON object of the lensflare at index position `i`.
- `remove(/int/ i)`, Removes the lensflare at index position `i`.
- `insert(/int/ i, /Object/ h)`, Inserts a lensflare `h` as JSON object (see also chap. 4.2 (→ p.21)) at index position `i`.

```
/Object/ location()
```

Returns the location object, if defined. See chap. 4.2 (→ p.21).

```
lock()
```

Locks the Viewer for user input.

```
/Object/ map()
```

Returns the mapElement object, if defined. See chap. 4.2 (→ p.22).

```
/Number/ maxZoom()
```

Returns the maxzoom value as set in camera.

```
openPanorama(/*String*/ parameterFile,/*String*/ cmd,/*Boolean*/ blend,  
             /*Number*/ time,/*Boolean*/ nodeOnly)
```

Opens another panorama in the same Viewer window. For that purpose the parameter `parameterFile` must contain the name of the parameter file of the destination panorama.

The parameters `cmd`, `blend`, `time`, and `nodeOnly` are optional. With `blend` set to `true` the Viewer fades smoothly into the new panorama. The duration of the fade in seconds can be set with `time`.

The `cmd` parameter supports a simple list of commands to set the field-of-view for the new panorama. For that purpose you can set for a 3D panorama the parameters `view.pan`, `view.tilt`, and `view.hfov`. For 2D zoom images you have the parameters `view.x`, `view.y`, and `view.zoom` (please see also `setView`).

As an example let's assume we want to open a panorama with a view in the direction 100 degrees horizontally, 5 degrees vertically and a field-of-view of 45 degrees.

```
<input type="button" value="Open Panorama" onclick=  
  "document.getElementById('panoStudioViewerID').panoStudioViewer.  
    openPanorama('Panorama2.js','view.pan=100.0;view.tilt=5.0;view.hfov=45;');" />
```

If `nodeOnly` is `true`, then the Viewer opens just the content of the node element in the parameter file and keeps controls and other elements of the current panorama.

```
/*Number*/ panoType()
```

Returns 0, if it is a 3D panorama, 1, if it is a 2D zoom image.

```
panTo(/*Number*/ pan,/*Number*/ tilt,/*Number*/ hfov,/*Number*/ time,/*Number*/  
      delay,/*String*/ transition,/*Boolean*/ lock,/*Function*/ onComplete)
```

Pans the camera to another view with the destination parameters `pan`, `tilt`, `hfov`. In a planar zoom image the destination parameters are image center and zoom (`x`, `y`, `zoom`).

`time` specifies the pan duration, `delay` is the delay until the pan starts.

`transition` is the tween type. `easeInOutQuint` is the default setting (see also *tween types*, p. 40).

With `lock` you can block user interactions until the pan ends, otherwise a user interaction interrupts the pan. `onComplete` is an optional function which is called when the pan ends.

```
remove(/*String*/ id)
```

Find a layer based on its `id` and delete it.

```
/*Object*/ safeArea()
```

Returns an object with the values `left`, `top`, `right`, `bottom` in pixels for the indentation that is active when the Viewer is configured with the `viewport_fit_cover` option and it runs in fullscreen mode or fills the browser window. In this case the values correspond to the CSS `safe-area-inset-*` environment variables.

```
/*Object*/ screenToPano()
```

Returns an object with the angles (`pan`, `tilt`) of the viewing direction at the screen coordinates `x`, `y`. In a planar zoom image the object contains the `x`, `y` coordinates in the image in the interval `[0,1]`.

```
setView(/*Number*/pan,/*Number*/tilt,/*Number*/hfov)
```

Sets the viewing direction in the panorama to the angles `pan` and `tilt` and the horizontal field-of-view to the value `hfov`.

In a planar zoom image `setView` expects as parameters the center of the view and the zoom (`x`, `y`, `zoom`).

```
showCursor()
```

Shows the mouse cursor.

```
startAutoPlay()
```

Starts the auto play.

```
stopAutoPlay()
```

Stops the auto play.

```
/*String*/ systemInfo()
```

Provides browser and operating system information.

```
toggleFullscreen()
```

Switches between fullscreen and windowed mode.

```
/*String*/ tr(/*String*/ text)
```

Translates `text`. Uses the `translate` object (chap. 4.2 (→ p.25)). Returns the translation of the string `text`, if there is an according key value pair for the current language code of the browser available in the `translate` object, otherwise it returns `text`.

```
unlock()
```

Unlocks the Viewer for user input.

```
/*String*/ versionInfo()
```

Provides version information of the PanoramaStudio Viewer.

```
/*Number*/ width()
```

Current width of the Viewer in pixels.

```
/*Number*/ zoom()
```

Returns the current zoom value. Where 1.0 corresponds to a 100% view. Smaller values mean zoomed out, larger values mean zoomed in.

5.3.3 Local functions of layers

The common base of all visual elements in a panorama (buttons, logos, hotspots, textboxes, ...) is a so called layer object.

Each layer object provides the following functions:

```
setVisible(/*Boolean*/ visible)
```

Call with true to make the layer visible, else invisible.

```
updateAlign()
```

Call when you modified a position property (xoff, yoff, align, ...) to update the layer.

```
updateStyles()
```

Call when you modified a style property (style, stylehover, styleactive) to update the layer.

```
/*Object*/ get(/*String*/ id)
```

Find a layer or object based on its id.

```
globalToLocal(/*Object*/ coordinate)
```

Converts the x and y properties of the coordinate object from viewport coordinates to coordinates relative to this layer.

```
/*Number*/ offsetLeft()
```

Returns to relative left position of the layer on the viewport.

```
/*Number*/ offsetTop()
```

Returns to relative top position of the layer on the viewport.

```
/*Number*/ offsetWidth()
```

Returns to width of the layer on the viewport.

```
/*Number*/ offsetHeight()
```

Returns to height of the layer on the viewport.

```
tween(/*Object*/ parameters)
```

Use a tween to interpolate a variable smoothly between two values over a period of time. Allows to create animations.

The parameters object contains all parameters of the *Tween* as key/value pairs. The following parameters are supported:

- 'time', duration of the tween in seconds (default: 1.0)

- 'transition', tween type (default: easeOutExpo)
- 'delay', time until the tween starts (default: 0)
- 'onInit', a function called when the tween starts (default: null). The function gets as arguments the Json object of the layer and the object with the parameters as key/value pairs. This allows to dynamically modify the destination parameters when the tween is initialized (onInit(layer,parameters)).
- 'onUpdate', a function called on each update of the tweening function (default: null). The function gets as argument the Json object of the layer.
- 'onComplete', a function called when the tween ends (default: null). The function gets as argument the Json object of the layer.
- 'breakOnUser', boolean. If true, then each user interaction will immediately end the tween and onComplete will be called (default: false).

You can use all numeric properties of the layer's JSON object belonging to the tween as tween variables.

Example for a tween animation which fades out the controls (id: toolbar) by moving them 70 pixels to the bottom within 0.5s and moves them this way out of the Viewer window.

For that the tween interpolates smoothly the yoff property of the layer. The onUpdate function is used to apply the modification of the yoff variable to the display:

```
var o = this.get('toolbar');
if (!!o){
  o.tween({'yoff': -70, 'time': 0.5, 'onUpdate': function(a){ a.updateAlign(); } }
);
```

The following tween types (also called *easing functions*) are supported:

Linear:

- easeLinear

Ease In:

- easeInQuad
- easeInCubic
- easeInQuart
- easeInQuint
- easeInSine
- easeInExpo
- easeInCirc
- easeInElastic
- easeInBack
- easeInBounce

Ease Out:

- easeOutQuad
- easeOutCubic
- easeOutQuart
- easeOutQuint
- easeOutSine
- easeOutExpo

- easeOutCirc
- easeOutElastic
- easeOutBack
- easeOutBounce

Ease In/Out:

- easeInOutQuad
- easeInOutCubic
- easeInOutQuart
- easeInOutQuint
- easeInOutSine
- easeInOutExpo
- easeInOutCirc
- easeInOutElastic
- easeInOutBack
- easeInOutBounce

Ease Out/In:

- easeOutInCubic
- easeOutInQuart
- easeOutInQuint
- easeOutInSine
- easeOutInExpo
- easeOutInCirc
- easeOutInElastic
- easeOutInBack
- easeOutInBounce

A visualization of the different tween types can be found e.g. here:
<http://easings.net/en>

5.3.4 Local functions of buttons

Besides the inherited layer functions (chap. 5.3.3, p.40) a button provides also the following functions:

```
updateSkins()
```

Call when you modified a skin property (skin, skinhover, skinactive) to update the button.

5.3.5 Local functions of textboxes

Besides the inherited layer functions (chap. 5.3.3, p.40) a textbox provides also the following functions:

```
updateText(/*String*/ htmlText)
```

Changes the content of the textbox. Can contain arbitrary HTML.

5.3.6 Local functions of audio objects

Besides the inherited layer functions (chap. 5.3.3, p.40) an audio object provides also the following functions:

```
play()
```

Starts playing the audio.

```
pause()
```

Pauses the currently playing audio.

```
/*Boolean*/ isAudioPlaying()
```

Returns true, if the audio object is currently playing.

```
/*Boolean*/ isPlaying()     /*deprecated*/
```

Returns true, if the audio object is currently playing.

5.4 Properties

You can access and modify all properties of the panoStudioViewer instance defined in the parameter file at runtime with JavaScript (see chap. 5.1 (→ p.29)). Similarly, you can access and modify all properties of layer objects with `get(...)` (see chap. 5.3.2 (→ p.36)).